NEC

PC-100 Series

N100-BASIC リファレンスマニュアル

PC-10000-BM2
PTS-258

リファレンスマニュアル

PC-100 Series

N_{100}-BASIC

NEC

ご注意
(1)本書の内容の一部又は全部を無断転載することは禁止されています.
(2)本書の内容に関しては将来予告なしに変更することがあります.
(3)本書は内容について万全を期して作成いたしましたが，万一御不審な点や誤り，記載もれなどお気付きのことがありましたら御連絡下さい.
(4)運用した結果の影響について(3)項にかかわらず責任を負いかねますので御了承下さい.
(5)乱丁，落丁は，お取替いたします.

XENIXは米国マイクロソフト社の登録商標です.
MSおよびMicrosoft(マイクロソフト)のロゴは米国マイクロソフト社の商標です.

目　次

文法編

リファレンス編

ご注意

1. ディスクファイルをOPENした場合は，END文，CLOSE文あるいはRESET文によってすべてのファイルを閉じるまで，ディスクをドライブから抜いたり，電源を切ったりしないようにしてください．このようなことをすると，ファイルが破壊されてしまいます．

2. プログラムの中で，同名のラベルを重複して使用すると，〝Duplicate label〟エラーが生じますが，いったんこのエラーが生じると，以後大文字でのコマンド入力はすべて同じエラーを起こしてしまいます．このようなときは，小文字の入力によって重複したラベルを削除するか，プログラム全体を削除(newコマンドによる)すればもとの状態に戻ります．

3. ディスプレイ画面に対してエスケープシーケンスを使用した場合，スクリーンエディタが正常に動作しなくなることがあります．このようなときは，SCREEN文，WIDTH文あるいはWIDTH USING文によって，現在の画面モード以外のモードを選択すればスクリーンエディタは正常状態に復帰します．ただし，これによっても正常にならないときには，BASICを再起動しなくてはなりません．

4. SHELLコマンドにより画面上に出力されたデータや文字に対してはスクリーンエディタによる編集を加えることはできません．またSHELLコマンドの実行後，画面上に以前より残っているBASICのプログラム行やコマンド行に対してスクリーンエディタによる編集を加えても，正しく修正できないことがあります．このようなときには，いちど画面をクリアすればスクリーンエディタは正常状態に戻ります．

N_{100}-BASIC
REFERENCE MANUAL

文法編

1 N_{100}－BASICの特徴

MS-DOSの下で走るN_{100}-BASICは，DOSの特性を存分に生かして，従来のBASICでは実現できなかった種々の強力なコマンドをサポートしています．以下にその特徴を述べます．

1．XENIX* スタイルのDOSインターフェイスが，使いやすいオペレーティング環境を提供します．

2．標準入力，標準出力を任意に変更することのできる，リダイレクト機能を有しています．

3．キャラクタデバイスをサポートしていますので，ユーザーがインストールしたデバイスを使用することができます．

4．DOS上でサポートされるディレクトリの階層構造（木構造）をBASICでそのまま利用できます．したがって広大なディスク空間を，わかりやすく，しかも効率よく構成することができますので，データの管理やメンテナンスを非常に楽に行うことができます．

5．SHELL機能により，現在のBASICの状態を離れることなく，DOSのコマンド（内部コマンドも外部コマンドも含みます）を走らせることができます．走らせた後，制御をBASICに戻せば，以前のBASICの状態は，すべて保たれています．例えば，テキストエリアにロードされているBASICのプログラムはそのまま残っており，変数の値なども前のまま保存されています．この機能を使えば，BASICレベルにいながら，DOSのSORTコマンドを呼び出してデータのソートを行ったり，CHKDSKコマンドによってディスクのチェックを行ったりすることが簡単にできます．

6．割込み機能が拡張され，強力になっています．タイマ割込みをはじめ，ユーザー定義の可能なキーボード割込み，マウスによる割込み，コミュニケーション回線RS-232Cによる割込みなど，各種サポートされています．

7．10進演算パックの採用により，BASICの演算はすべてBCD（2進化10進）コードで行われます．有効桁数は，単精度ならば6桁，倍精度ならば14桁まで保証されます．BCD演算ですから，誤差の少ない，高精度な演算結果を得ることができます．

8．DOS上で日本語がサポートされていることにより，BASIC上においても簡単に日本語の処理を行えます．とくに，漢字の入力には，カナ漢字変換を利用することができ，簡単に，速く文字を打ち込むことができます．

9．手軽なポインティングデバイスとして有用なマウスを，BASICで利用することができます．非常に簡単に画面上をドット単位でアクセスすることが可能なマウスを使えば，簡便な操作で複雑な処理を行うことのできるアプリケーションプログラムをつくることができます．

10．ディスプレイ画面を，横長方向にでも，縦長方向にでも表示させることができます．

＊XENIXシステムは，米ベル研究所が開発したUNIX®オペレーティングシステムを米マイクロソフト社が改良，拡張した会話型のオペレーティングシステムです．

11. 大容量のグラフィックス専用 RAM を直接制御できるため，高速な，720×512の高分解能グラフィックスを実現できます．

12. 強力なグラフィックコマンドがサポートされています．例えば，ワールド座標とスクリーン座標を自由に操作できる WINDOW，VIEW，PMAP などのステートメントや関数，スクリーン座標に相対指定で図形を描くグラフィックマクロ命令をサポートする DRAW 文，グラフィック図形を瞬時に記憶･再現できる GET@，PUT@の各文など，便利でパワフルな種々のコマンドがあります．

13. グラフィックス表示では，カラーボードとカラー CRT の使用（model 30 では標準）により，1ピクセル（画素）につき512色の中から選んで色をつけることができます．同一画面内では，512色の中から16色を選んで表示できますから，多彩な表現が可能になります．

14. SOUND 文，PLAY 文により，いろいろな高さの音を出すことができます．とくに PLAY 文はミュージックマクロ命令と呼ばれる音楽専用のコマンドを文字列で与えることにより，いろいろな曲を演奏することができます．このミュージックマクロ命令は，楽譜イメージの数種のコマンドから成り，簡単に音符を音として表現できるようになっているものです．

15. 英数字の表示には，標準体，ボールド体，イタリック体の3種類の文字体を選んで使用することができます．

2 動作モード

BASIC を起動すると(起動方法については N_{100}-BASIC ユーザーズガイドを参照してください)，PC-100は，画面に"Ok"という文字を出力し，BASIC のコマンドを入力できる状態になります．この状態のときは，あらゆる BASIC のコマンドならびにステートメントをキーボードから入力できます．

ダイレクトモード

行番号をつけずに BASIC の文法に沿った文を入力した場合，その文は，リターンキーを入力後，すぐに実行されます．これをダイレクトモードにおける実行といいます．

プログラムモード

行番号（0～65529）をつけて文を入力した場合，それはメモリの中にプログラムとして行番号とともに格納されます．そしていったん格納されたプログラムは，RUN コマンドおよび GOTO 文，GOSUB 文によって実行させることができます．これをプログラムモードによる実行と呼びます．

3 文

文とは，BASIC が行う手続きを記述している最小単位です．

文には，BASIC が実行する式，ステートメント，コマンド，関数などを書くことができます．また文をコロン（：）を用いて区切ることにより，1行中に2つ以上の文を書くことができます．これは複文（マルチステートメント）と呼ばれ，複文は，1行（行番号も含めて）255文字以内（日本語文字が入っている場合は異なる）の長さまで許されています．

4 行番号

BASIC の各プログラム行は，行番号で始まらなければなりません．行番号には，0～65529までの整数を用います．行番号はプログラム行をメモリに格納する順序を示し，実行も行番号の若い方から行われます．また分岐や編集の目印としても使用されます．

行番号の代わりにピリオド(.)を使うことができる場合があります．ピリオドは LIST, AUTO, DELETE, EDIT などのコマンド中で，エラー発生，編集などによって BASIC 内のポインタが示している現在の行を表すものです．

```
LIST .
AUTO .
DELETE .－100
```

5 ラベル

BASIC では，プログラムの流れを変える命令として“GOTO”，“IF THEN”などを使います．N_{100}-BASIC ではこれらの飛び先の指定を行番号で行うことも，ラベルで行うこともできます．

次のプログラムは行番号を使って飛び先を指定した例です．

```
10 INPUT A
20 IF A<0 THEN 80
30 IF A>0 THEN 60
40 PRINT "zero"
50 GOTO 90
60 PRINT "plus"
70 GOTO 90
80 PRINT "minus"
90 END
```

これは，10行で入力された値が正か負かゼロかをしらべるプログラムです．ここで処理の流れを変える命令が20，30，50，70行に使われていて，その飛び先の指定はすべて行番号になっています．

ラベルを使って先程のプログラムを書き換えてみると次のようになります．

```
10 START
15       INPUT  A
20       IF  A< 0   THEN  MINUS
25       IF  A> 0   THEN  PLUS
30       PRINT   "zero"
35       GOTO  EXIT
40 PLUS
45       PRINT   "plus"
50       GOTO  EXIT
55 MINUS
60       PRINT   "minus"
65 EXIT: END
```

このようにラベルは，ユーザーが分岐先の目印として独自に付けることのできる名前です．

ラベルを使うときは，次の注意を守らなければなりません．

1．ラベルの先頭の文字は必ず英文字またはアスタリスクにします．

2．ラベルに使用できる文字は，英文字，数字，ピリオド(.)で，大文字と小文字の区別はありません．アスタリスク(*)は，ラベルの先頭以外には使えません．

3．BASICの予約語(PRINTなど)をラベルとして使用することはできません．ただし，中に含む場合はかまいません．

4．ラベルの長さは，プログラムの1行中に書ける範囲（255文字）内ならば何文字でもかまいません．

5．参照されるラベル（呼ばれる方のラベル）は，必ず行の最初になければなりません．

6．1行中でラベルの後に続けて命令を記述し，マルチステートメントとするときは，コロン(:)またはスペースによって区切ります．

以上のような制限を無視すると "Syntax error" となります．

その他参照されるラベルに同じものがあった場合，2重定義されているという意味の "Duplicate label" エラーが生じます．これらのエラーの検出は，他のエラーメッセージの場合と異なり，"RUN"したときプログラムの実行に先だって行われますから，ラベルのエラーがあるとプログラムを1ステップも実行することができません．

また，この場合のエラーメッセージはエラー個所を示す〈行番号〉は伴いません．

ラベルは，プログラム中で分岐の目印として使用する他，コマンドレベルで"LIST"，"DELETE"など〈行番号〉を対象とするパラメータにはすべて使用することが可能です．

例）
```
LIST  START-LAST
DELETE  *LOOP1-*EXIT1
```

なお，いったん"Duplicate label"エラーが生じますと，以後大文字でのコマンド入力はすべて"Duplicate label" エラーを生じてしまいます．このエラーが出たら，以後のコマンド入力はすべて小文字で行ってください．なお，重複したラベルを消去するか，new コマンドを入力すれば，もとの状態にもどります．

6 使用できる文字とコントロールキャラクタ

N_{100}-BASIC で使用できる文字は，英文字，カナ，数字，特殊記号，そして日本語文字ならびにグラフィックキャラクタより構成されており，これらはキャラクタセットと呼ばれます．英文字は大文字と小文字で，数字は 0 から 9 までです．これらキャラクタセットの詳細は，付録の〝ASCII コード表〟および〝漢字コード表〟を参照してください．

また，これとは別に，スクリーンエディタに直接指示を与えたりするためのコントロールキャラクタがあります．これについては PC-100 BASIC ユーザーズガイドを参照してください．

7 特殊記号の使い方

BASIC では，演算子（+，−，＊，／）などの他にも特別な意味を持つ記号があります．ここでその意味をまとめて説明しておきます．

1．ピリオド（.）

現在 BASIC が着目している行番号の値を持っておりポインタとして使用することができます．BASIC が着目しているとは，新しい行を挿入した，エラーが発生したなどの行です．

例）　LIST .

2．マイナス（−）

LIST, DELETE 命令など行の範囲を指定するとき，何行から何行までという場合に使います．演算子のマイナス記号（−）と同じ記号です．

例）　DELETE　100−200

3．コロン（:）

マルチステートメントの区切りとして使います．

例）　A=B+C : PRINT　A

4．カンマ（,）

PRINT，INPUT などパラメータが並ぶ場合その区切りとして使います．

例）　INPUT　A，B，C

　　　SCREEN　1，0，0，1

5．セミコロン（;）

PRINT 文などの区切りとして使います．

例）　PRINT　〝A=〟; A

6．アポストロフィ（'）

REM 文（リマーク）の代用として使います．

7．クエスチョンマーク（?）

PRINT 文の代用として使います．

8．スペース

BASIC は原則としてスペースを無視します．ただしコマンド，ステートメントの後には必ずスペースを入れなければなりません．

8 予約語

BASICは，コマンド，ステートメント，関数，システム変数，演算子などの名前を，処理上，特殊なものとして扱います．これらは〝予約語〟と呼ばれ，文字どおりBASICによってその使用方法を予約されているものです．したがって，予約語を変数名やラベルとして使用したりすることはできません．

予約語を使用するにあたっては，BASICがその語が予約語であるか否かを認識できるようにするため，語の前，後，またはその両方に，BASIC文法上で許された，あるいは規定された特殊文字(スペース，ダブルクォーテーション（〝），ナンバー記号（#），コロン（：）など）を挿入しなくてはなりません．

BASICのすべての予約語は，付録の〝予約語〟に掲げてあります．

9 定数

定数の種類

N_{100}-BASICの定数の種類は次のとおりです．

- 定数
  - 文字定数（漢字なども含む）
  - 数値定数
    - 整数型
      - 10進形式
      - &O（8進）形式
      - &H（16進）形式
      - &J（JISコード）形式
      - &K（区点コード）形式
    - 実数型
      - 単精度型
        - 固定小数点形式
        - 浮動小数点形式
      - 倍精度型
        - 固定小数点形式
        - 浮動小数点形式

文字定数

文字定数とはダブルクォーテーション(″)で囲まれた英数字，カナ文字，記号ならびに漢字などの255文字以下の列のことです．なお，ダブルクォーテーション(″)を文字列の中に記述する場合は，CHR$(34)を用いなければなりません．

例）　〝Good Morning〟
　　　〝1 2 3 4 5 6 7 8 9〟
　　　〝パーソナル　コンピュータ〟
　　　CHR$ (34)＋〝ABC〟＋CHR$ (34)
　　　〝日本電気〟

数値定数

数値定数は，整数型，実数型に分けられ，それらは正あるいは負の数，または 0 です．
負の数の前には必ず負記号をつけなければなりませんが，正の数の前の正記号は省略することができます．

整数型

(1) 10進形式

－32768から＋32767までのすべての整数．または，数の後に％をつけたもの．小数点をつけることはできません．

例)　32767

　　－123

　　32767%　←整数を表す．

(2) 8 進形式

頭に＆Oまたは＆を伴った，8 進数（0 から 7 まで）の数字の並び．

＆0～＆177777の範囲

例)　＆12345

　　&O7777

(3) 16進形式

頭に＆Hを伴った，16進数（0 から F まで）の並び．

＆H0～＆HFFFF の範囲

例)　&H100

　　&HCFFF

(4) JIS コード形式

頭に＆J を伴った，16進数（0 から F まで）の並び．＆J 以下に書かれた JIS16 進コードに対応する内部連続コードの値をもちます．内部連続コードについては BASIC ユーザーズガイドをご覧ください．

例)　&J304E

　　&J4D7D

(5) 区点コード形式

頭に＆Kを伴った，10進数の並び．＆K以下に書かれた JIS 区点コードに対応する内部連続コードの値をもちます．内部連続コードについては BASIC ユーザーズガイドをご覧ください．

例)　&K1646

　　&K4593

注意：どの形式を用いて書き表された数値でも，PRINT，LPRINT 文などを使用した場合は，すべて10進形式で出力されます．

実数型

実数型は，精度の観点から単精度型と倍精度型に，また表現方法の観点から固定小数点形式と浮動小数点形式に分けられます．

単精度型

有効桁６桁の精度で格納され，６桁以下で表示されます．扱える数値は，−9.99999Ｅ＋62〜9.99999Ｅ＋62です．

（１） ６桁以下の実数

（２） Ｅを使った指数形式

（３） 最後に！を伴った数

例）
1.23
−7.09Ｅ−06
3525.68
3.14！

倍精度型

有効桁14桁の精度で格納され，14桁以下で表示されます．扱える数値は，−9.9999999999999Ｅ＋62〜9.9999999999999Ｅ＋62です．

（１） ７桁以上の数

（２） Ｅを使った指数形式

（３） 最後に#を伴った数

例）
1234567890
−1.09432Ｅ−06+0.3141592653Ｅ＋01
56789.0#
8657036.1593976

固定小数点形式

正または負の，小数点を含む数．一般の小数点数はすべてこの形式です．

例）
123.456
−.26839696823

浮動小数点形式

正または負の，指数形式で表示された数．即ち，〝仮数部〃＋〝Ｅ〃＋〝指数部〃という形で表現される，科学計算などでよく用いられる形式の数です．

例）
1.4343Ｅ＋20
−2.22333444555Ｅ−99

10 変数

BASICのプログラム中で使われる値を格納するためのエリアに，英数字から成る名前を対応させたものが変数です．変数にも，定数と同様，数値変数と文字変数があります．

変数の値は，プログラマによって定義され，演算，参照などに使うことができます．これは演算，代入実行後，割り当てられます．変数は，値が割り当てられるまえに参照されると，数値変数ならば0が代入され，文字変数ならばヌルストリング（空の文字列）が代入されます．

変数名と型宣言文字

変数名は，最大40文字の，英数字とピリオド(.)の組合せで表されるもので，先頭の文字は英字でなくてはなりません．また，漢字，ひらがな，カタカナは，変数名に使うことはできません．

BASICでは，変数名の組合せが少しでも異なると違う変数として扱われます．例えば，次の2つは区別されます．

例1）　ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890A
　　　　ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890B

変数名は予約語（BASICで使われるコマンド，ステートメントなどのキーワード）であってはなりませんが，予約語を含んだものはかまいません．ただし，FNで始まる変数名は許されません．また英文字において大文字，小文字の区別はありません．

例2）　RUN　←変数とはならない
　　　　RUN1 ┐
　　　　　　　├←変数として使える
　　　　XRUN ┘

同じ変数名でも，違う型宣言文字が付加されていれば区別されます．型宣言文字は変数名の最後につけて，その変数の型を表します．型宣言文字を省略すると，〝#〟がついているとみなされます（倍精度実数型変数となる）．

型宣言文字
- % 整数型
- ! 単精度実数型
- # 倍精度実数型
- $ 文字型

例3）　A# ┐
　　　　A% ├これらは区別されるが，AとA#は同じとして扱われる．
　　　　A$ ┘

変数の型を宣言するのにもう1つ便利な方法があります．それはDEFINT，DEFSNG，DEFDBL，DEFSTRの型宣言文をプログラム中で用いる方法です．これについてはリファレンス編のDEFINT／SNG／DBL／STRの項をご覧ください．

配列変数

配列変数とは，１つの変数名でいくつかの要素を参照することのできる変数です．配列変数のそれぞれの要素は，整数または整数表記による添字によって参照されます．

配列変数の型は，他の変数と同様，型宣言文字あるいは型宣言文を用いて指定します．

配列変数の次元は255次元まで，添字はメモリ容量の許す限り使うことができ，これらの大きさはDIM文（リファレンス編参照）で宣言します．添字は原則として０から始まります（添字の下限が０である，といいます）から，実際の要素数は添字の数＋１となります．

例）

```
DIM  A(10)              1次元配列，要素の数＝11
DIM  T#（10，50）        2次元配列，要素の数11×51＝561
DIM  ME$（2，5，3）      3次元配列，要素の数3×6×4＝72
```

OPTION BASE 文(リファレンス編参照)を用いて添字の下限を１にすることもできます．この場合，実際の要素数は添字の数と同じとなります．

なお，添字の数が10以下で，３次元以下の配列の時は DIM 文による宣言を省略することができます．

11 型変換

N_{100}-BASIC の数値データは，必要に応じてその型から他の型に変換することができます．ただし，文字型と数値型の間では，VAL 関数や STR$関数を用いない限りこのような変換を行うことはできません．

（1）ある型の数値データが，違った型の数値変数に代入された場合，数値は，その変数名によって宣言された型に変換されます．

例）

```
10  ABC%=1.234
20  PRINT  ABC%
RUN
1
```

（2）精度の違う数値間の演算の場合，精度の高い方に変換されて，演算が行われます．たとえば，10#／3の場合，10#／3#として演算が行われます．

例）

```
10  A#=10#／3%
20  B#=10#／3#
30  PRINT A#, B#
RUN
3.3333333333333     3.3333333333333
```

（3）論理演算の場合，扱われる数値はすべて整数に変換され，結果は整数で与えられます．

例）

```
10  A#=12.34
20  B=NOT A#
30  PRINT B, A#
RUN
-13          12.34
```

（4）実数が整数に変換される場合は，小数点以下は四捨五入されます．このとき，整数型で扱える範囲を越えた場合はエラーが起こります．

例）

```
10  A%=34.4
20  B%=34.5
30  PRINT  A%, B%
RUN
34            35
```

```
10 A#=1.234E+07
20 B%=A#
30 PRINT B%, A#
RUN
Overflow in 20
```

（5）倍精度変数が単精度変数に代入された時は，変数の値は有効数字6桁に丸めたものとなります．

例）

```
10 A#=1.23456789#
20 B!=A#
30 PRINT A#, B!
RUN
1.23456789       1.23457
```

12 式と演算

式とは定数や変数を演算子で結合した一般的な数式や，単に文字や数値，あるいは変数だけのものをいいます．

例）

```
"BASIC"
3.14
10+3/5
A+B/C-D
TAN (DO)
```

BASICの演算は次の5つに分類されます．

1．算術演算
2．関係演算
3．論理演算
4．関数
5．文字列演算

算術演算

算術演算子には次のようなものがあります．

実行順	演算子	演算	例
↓	^	指数演算	X ^ Y
	−	負記号	−X
	*，/	乗算，実数の除算	X * Y，X/Y
	+，−	加算，減算	X+Y，X−Y

次に式の例を示します。

	代数表記	BASIC の表記
1）	$2X+Y$	`2 * X+Y`
2）	$\frac{X}{Y}+2$	`X／Y+2`
3）	$\frac{X+Y}{2}$	`(X+Y)／2`
4）	X^2+2X+1	`X ^ 2+2 * X+ 1`
5）	Y^{Y^2}	`X^ (Y ^ 2)`
6）	$(X^Y)^2$	`X ^ Y ^ 2`
7）	$Y(-X)$	`Y * -X`

（1） 整数の除算と剰余の計算

整数の除算は¥によって行われます．扱われる数値が実数の場合は，演算が実行される前に小数点以下が四捨五入されます．商は小数点以下が切り捨てられた整数となります．

例） 10¥3＝3　　（10／3＝3・・・1）

23．75¥5＝4　　（24／5＝4・・・4）

剰余の計算は MOD によって行われます．扱われる数値が実数の場合は，演算が実行される前に小数点以下が四捨五入されます．結果は整数の割り算の余りです．

例） 13.3 MOD 4＝1　　（13／4＝3・・・1）

25.68 MOD 6.99＝5　　（26／7＝3・・・5）

（2） 0での除算

式の実行時に 0 での除算が行われた場合は，〝Division by zero〞エラーを出力しますが，結果は計算機が扱うことのできる最大の数とみなし，それを除算の結果として処理を続行します．

また，べき乗の実行時に，0 に対して負のべき乗を行った場合も同様になります．

例）
```
PRINT  2／0
Division by zero
  9.9999999999999 E +62

PRINT  0 ^ - 1
Division by zero
  9.9999999999999 E +62
```

（3） 桁あふれ（オーバーフロー）

代入や演算の結果がその変数の型内で表現することのできる範囲をこえた場合，桁あふれが発生します．

桁あふれが起こった場合，BASIC は〝Overflow〞エラーを出力し，BASIC が扱うことのできる最大の結果として与え，処理を続行します．

例）
```
PRINT  3 ^ 300
Overflow
  9.9999999999999 E +62
```

関係演算子は2つの数値を比較するときに用います．結果は，真（－1），偽（0）で得られ，条件判定文などプログラムの流れを変えるのに用いられます（IF 文参照）．

関係演算子	内容	例
＝	等しい	X＝Y
＜＞,＞＜	等しくない	X＜＞Y，X＞＜Y
＞	小さい	X＜Y
＜	大きい	X＞Y
＜＝,＝＜	小さいか等しい	X＜＝Y，X＝＜Y
＞＝,＝＞	大きいか等しい	X＞＝Y，X＝＞Y

注意：＝は代入文（LET 文）にも使うので注意すること．

IF 文の中での使い方の例を示します．

```
IF  X=0  THEN  1000
IF  A+B<>0  THEN  X=X+1 : Y=Y+1
```

論理演算

論理演算子は複数の条件を調べたり，ビット操作やブール演算を行ったりするのに用います．論理演算は，ビットごとに0または1を結果として与えます．

各論理演算の内容を次に示します．

XOR ＝ exclusive or（排他的論理和）

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

NOT ＝ not（否定）

X	NOT X
1	0
0	1

IMP ＝ implication（包含）

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

AND ＝and（論理積）

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

EQV ＝ equivalence（同値）

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

OR ＝ or（論理和）

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

論理演算子も関係演算子のように，プログラムの流れを変えるのに用いられます．この場合，論理演算子は２つ以上の関係演算子と結ぶことができます．

例)　(１) IF　X< 0　OR　99<X　THEN　1000
　　(２) IF　0 <X　AND　X<100　THEN　X= 0
　　(３) IF　NOT (A= 0) THEN　20
　　(１) Xが負または，99より大きれば，行番号1000へ飛ぶ．
　　(２) Xが正でかつ，100より小されば，X＝０を代入する．
　　(３) Aが０でなければ，行番号20へ飛ぶ．

注意：論理演算子は演算の前に，扱う数値を-32768から＋32767までの２の補数表示の整数に変換します．もし，この範囲外となれば〝Overflow〞エラーとなります．０（偽）または－１（真）しか与えられなかったなら，論理演算子または０または－１しか与えません．

指定された論理演算では，この整数に対しビットごとに演算を行います．したがって，論理演算子はバイトデータをあるビットパターンに照らし合わせて調べることができます．

たとえば，AND演算子は機器のI／Oポートのステータスバイトの必要なビット以外のすべてのビットをマスクするのに使えます．またOR演算子はある２進数を作るために２つのビットパターンを混ぜ合わすことができます．

以下の例は論理演算子がどのように働くかの例です．

－1　OR　0 =－1	－1 =	(1111　1111　1111　1111)$_2$
	0 =	(0000　0000　0000　0000)$_2$
	－1　OR　0 =	(1111　1111　1111　1111)$_2$
48　AND　24=16	48=	(0000　0000　0011　0000)$_2$
	24=	(0000　0000　0001　1000)$_2$
	48　AND　24=	(0000　0000　0001　0000)$_2$
15　XOR　60=51	15=	(0000　0000　0000　1111)$_2$
	60=	(0000　0000　0011　1100)$_2$
	7　XOR　60=	(0000　0000　0011　0011)$_2$
17　EQV　12=－30	17=	(0000　0000　0001　0001)$_2$
	12=	(0000　0000　0000　1100)$_2$
	17　EQV　12=	(1111　1111　1110　0010)$_2$
28　IMP　9 =－21	28=	(0000　0000　0001　1100)$_2$
	9 =	(0000　0000　0000　1001)$_2$
	28　IMP　9 =	(1111　1111　1110　1011)$_2$
NOT　23　=－24	23=	(0000　0000　0001　0111)$_2$
	NOT　23=	(1111　1111　1110　1000)$_2$

関数

関数とは与えられたある引数に対して，ある決まった演算を行うもので，値としては，この演算の結果を返します．

BASICは〝組込み関数〟としてSIN（正弦），SQR（平方根）などの数値関数やCHR$，LEFT$などの文字列関数を本体にもっています．

また，BASICは〝ユーザー定義関数〟としてユーザーが自由に定義できる関数機能ももっています．これら2章の〝DEF FN〟の項で説明します．

また，これらの初等関数（SIN関数など）は，引数が単精度のときでも倍精度のときでも，返す値は倍精度となります．

一般に引数に整数しかとらないものは，小数部分を四捨五入して整数に丸めてから演算を行います．

次にその使い方の例を示します．

```
A=SIN (3.14)+COS (3.14)
PRINT  2, 2 * 2, SQR (2)
```

文字列演算

文字列の連結

文字列は演算子＋によって連結することができます．

例）
```
10 A$= "NEC" : B$= "PERSONAL" : C$= "COMPUTER"
20 D$=A$+ " "+B$+ " "+C$
30 PRINT D$
RUN
NEC  PERSONAL  COMPUTER
```

文字列の比較

文字も数値の比較に用いられるものと全く同じ関係演算子を用いて比較することができます．

=，<，>，<>，><，<=，=<，>=，=>

文字列の場合それぞれの文字列の最初から1文字ずつ，文字の比較を行います．もし相互に全く同じ文字列の場合は，その2つの文字列は等しくなりますが，1箇所でも違った場合は，その文字のキャラクタコードの大きい方の文字列が大きくなります．

キャラクタコードの大小は，英字のときはアルファベット順，カタカナのときはアイウエオ順，漢字のときはJISコード順に評価されます．つまり，すべてのキャラクタは，内部連続コードの並びに従って評価されるのです．

文字列の比較においては空白なども意味をもちますから注意してください．

以下の例では，結果はすべて真（－1）となります．

例）
```
"AA"<AB"
"BASIC"= "BASIC"
"X$"> "X #"
"PEN "> "PEN"
"cm"> "CM"
"DESK"< "DESKS"
"ABD"> "ABCDE"
```

このように，文字列の比較は文字列の内容を調べたり，文字をアルファベット順に並べたり（ソート）することに使うことができます．

13 演算の優先順位

演算は次の順位によって行われます．

1．カッコで囲まれたもの
2．関数
3．指数（べき乗）
4．負記号（－）
5．＊，／
6．￥
7．MOD
8．＋，－
9．関係演算子（<，>，＝など）
10．NOT
11．AND
12．OR
13．XOR
14．IMP
15．EQV

14 エラーメッセージ

N_{100}-BASICは，プログラムの実行を中断させなければならないようなエラーを実行時に検出した時，エラーメッセージを表示しコマンドレベルに戻ります．
ダイレクトモードでのエラーメッセージの書式は，

XXXX・・・

プログラムモードの書式は，

XXXX・・・in yyyyy

XXXX・・・はエラーメッセージで，yyyyyはエラーが検出された行番号です．エラーメッセージの内容については，付録の〝エラーメッセージ〟を参照してください．

15 画面モード

N_{100}-BASICは画面表示を白黒モードやカラーモードにしたり，また縦長方向表示や横長方向表示を選んだりすることができます．これらのモードの切り換えは，SCREEN文で行います．
なお，N_{100}-BASICはリストなどのキャラクタを出力する画面と，点や線などのグラフィックスを出力する画面を別の画面としてもつことはできません．

白黒モードとカラーモード

カラーディスプレイとカラーボード（model 30では標準装備）を使用している場合に限り，カラーモードを設定することができます．

白黒モードとカラーモードの指定は，SCREEN文の第1パラメータで行います．白黒モードの場合でもカラーモードの場合でも，画面は1枚だけです．したがって，白黒画面とカラー画面とを同時に使用することはできません．

白黒モードのときにPALETTE文を使用しますと，"Illegal function call"エラーとなります．また，このモードでCOLOR文を使用したり，CIRCLE文やLINE文などでパレット番号を指定したりした場合は，その指定が黒色でない限り文字やグラフィックスは画面上に白色で描かれることになります．

また，カラーボードがセットされていない場合は，白黒モードしか選択することはできません．PALETTE文を使用することもできません．この場合も，グラフィックコマンドでパレット番号を指定すると，黒色以外の色指定は，すべて白色で描かれます．

詳しくは，それぞれのグラフィックスのコマンドやステートメントの項をご覧ください．

横表示と縦表示

N_{100}-BASICは，ディスプレイ画面に対してキャラクタやグラフィックスを横長方向に表示したり，縦長方向に表示したりすることができます．この横表示と縦表示の指定は，SCREEN文の第2パラメータで行います．

漢字モードとグラフィックキャラクタモード

N_{100}-BASICは，漢字モードとグラフィックキャラクタモードを切り換えて使用することができます．この切り換えには，SCREEN文の第3パラメータを使います．グラフィックキャラクタモードでは，内部連続コードが128～255の範囲にあるキャラクタを，グラフィックキャラクタとして出力します．256以上のキャラクタは，漢字キャラクタにはなりませんから注意してください．

漢字モードでは，内部連続コードが256～4630の範囲にあるキャラクタを漢字キャラクタとして出力します．このとき，128～255の範囲のキャラクタは，グラフィックキャラクタとはなりません．

16 ディスプレイ画面上の座標系

N_{100}-BASICでは，キャラクタ画面とグラフィック画面とを明確に区別しているのではなく，基本的にはキャラクタもグラフィックスと同様な方法を用いて出力されます．ただ，1つ大きな違いは，キャラクタがスクリーンエディタによって編集可能な文字として認識されるのに対し，グラフィックスはスクリーンエディタに無視される点です．つまり，キャラクタは常にスクリーンエディタによって管理の可能な座標点に出力され，グラフィックスはこれとは無関係に自由な座標点に出力されるわけです．

次に，スクリーンエディタの管理下におかれるキャラクタ座標系と，そうではないグラフィック座標系について解説します．

キャラクタ座標系

普通，キャラクタを画面上に出力するためには，行（row）および桁（column）の概念が必要です．そしてまた，この概念はスクリーンエディタが編集箇所を正確に認識するためにも必要となります．N_{100}-BASICでは，WIDTH文，WIDTH USING文を用いてこの行と桁を指定することができます．この指定は，画面出力に対する指定であると同時にスクリーンエディタに対する指定ともなるわけです．

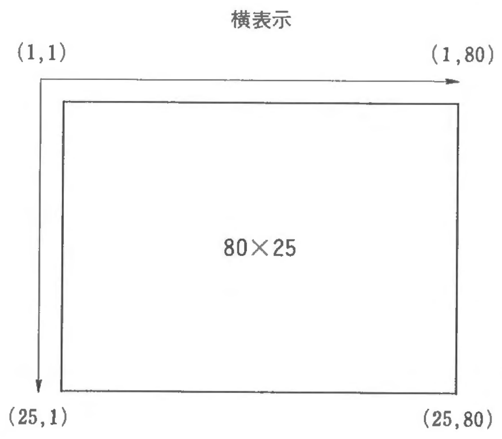
画面内に表示されるキャラクタの行数と桁数は，WIDTH文によっていろいろと変えることができますが，画面が横表示か縦表示かにより，つぎのような組合せが指定可能です．

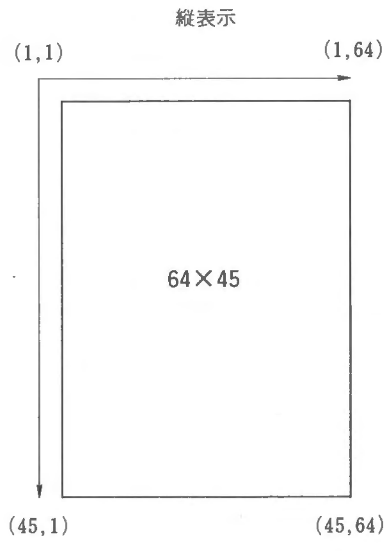
横表示（桁×行）	縦表示（桁×行）
80×25	64×45
80×32	85×45
80×64	85×90
90×25	—
90×32	—
120×25	—
120×32	—
120×64	—

ただし，WIHTH USING文を使用すれば，このような組合せ以外の表示方法を指定することもできます．詳しくは，WIDTH USING文の項を参照してください．

キャラクタ座標系の原点は，左上隅のカラムで，(1，1)で表されます．また，LOCATE文，SCREEN関数で参照されるキャラクタ座標は，（行，桁）つまり（Y，X）の順で表されます．CSRLIN関数は現在の行数（Y），POS関数は現在の桁数（X）を示します．

次に，キャラクタ座標系の例として，80×25モード（横表示）と，64×45モード（縦表示）を示します．



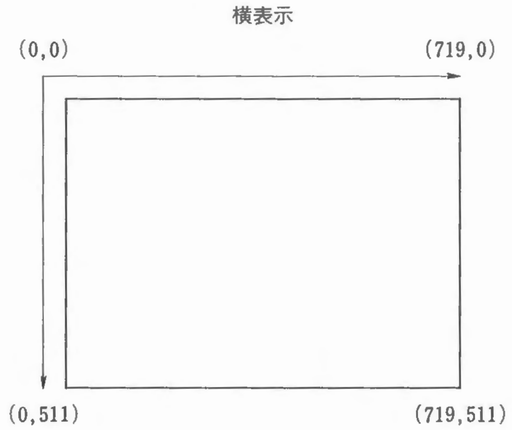


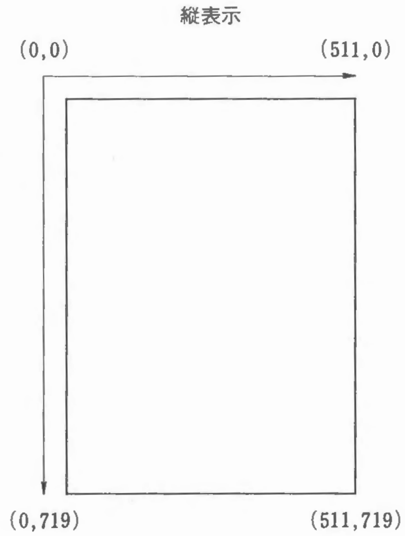
グラフィック座標系

N_{100}-BASICは，大容量のディスプレイ用メモリと高解像ディスプレイによって実現される高解像グラフィックスを完全にサポートします．

グラフィック座標系は，画面のモードに関係なく，常に720×512ピクセル(Pixel＝picture cell. 画素，ドットともいう)の広大な座標点を有し，その原点は左上隅の点で，(0，0)で表されます．なお，PSET，PRESET，LINE，CIRCLEなど各種のグラフィックコマンドで指定される座標点は，(横方向，縦方向)＝（X，Y）の順で表されます．キャラクタ座標系とは縦横の関係が逆になりますので，ご注意ください．

次に，グラフィック座標系を，横表示と縦表示のそれぞれの場合について示します．





17 ウィンドウとビューポート

ワールド座標系とスクリーン座標系

前項の"ディスプレイ画面上の座標系"の所で，N_{100}-BASICにはキャラクタ座標系とグラフィック座標系があることを説明しましたが，グラフィック座標系はさらにスクリーン座標系とワールド座標系という2つの座標系をもっています．

この2つの座標系は，グラフィック画面に対してのみ意味を持つもので，固定された画面上の座標系（物理的座標）だけでなく，仮空の大きな座標系（論理的座標）を扱うために必要なものです．

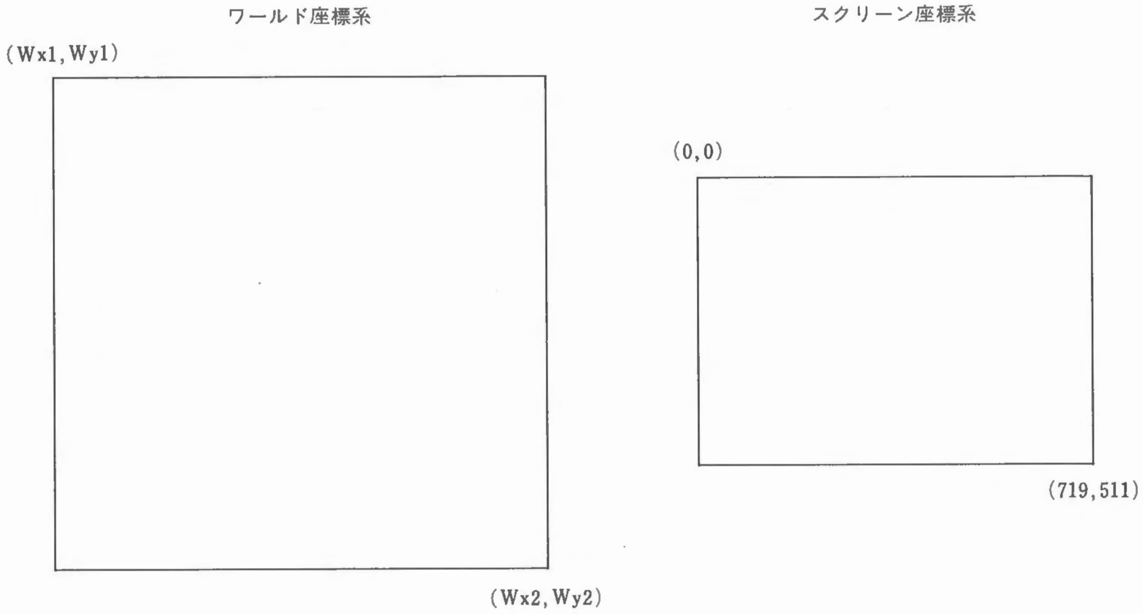
まず，スクリーン座標系とは，先程説明したグラフィック座標上に存在するもので座標上の各点は画面の1ピクセルに対応します．スクリーン座標については，ビューポートの項で説明します．

次にワールド座標系とは，ユーザーが使うことのできる最大の座標系であり，BASICが設定した仮空の座標です．その大きさは縦・横とも－9.99999E＋62～9.99999E＋62まで許されていて，負の領域座標も表現することができます．

ただし，水平方向の幅（W_X2-W_X1）及び垂直方向の幅（W_Y2-W_Y1）の絶対値が9.99999E＋62を超えてはなりません．

後に解説する，点を打つ(PSET)，線を引く(LINE)などのグラフィックコマンドのほとんどはこのワールド座標系に対して働きます．

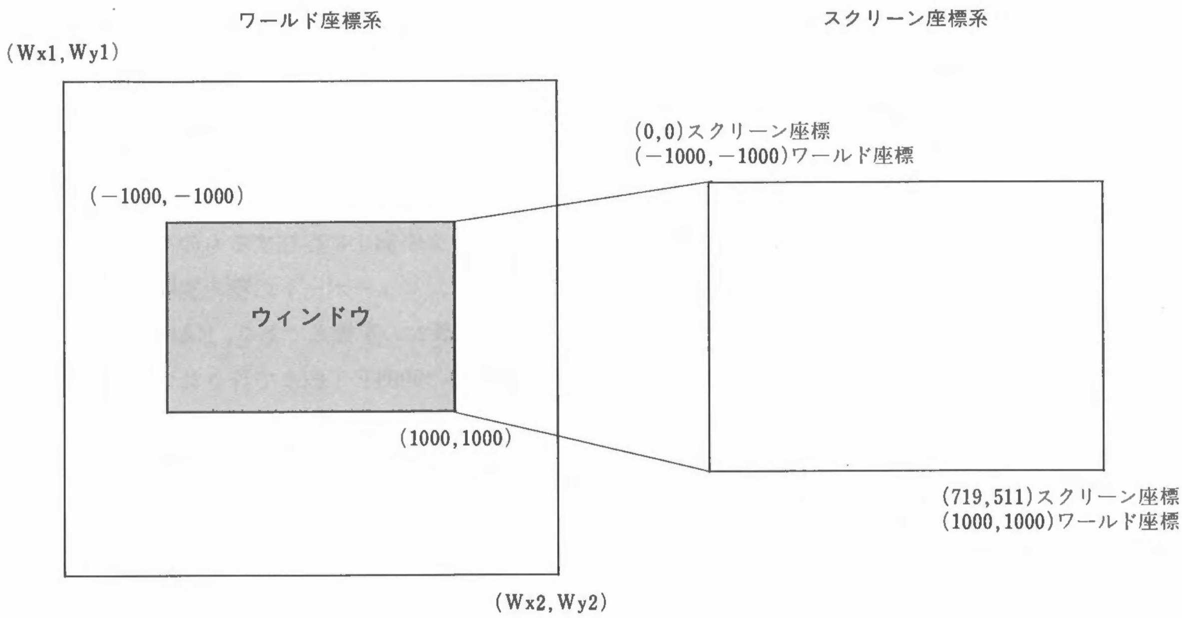
次の図はこの 2 つの座標系です.



ウィンドウ

前の項で説明したワールド座標系は，元々仮空のものであり，我々が実際に見ることのできる座標はあくまでスクリーン座標です.

従ってユーザーは，ワールド座標系の中から任意の領域を指定し，スクリーン座標上に表示領域として割り当てなければなりません. N_{100}-BASIC でこの操作をしているのが後述する WINDOW 文であり，この指定されたワールド座標上の領域のことをウィンドウ（のぞきまど）と呼びます. 次の図では，ワールド座標系の（−1000, −1000）から（1000, 1000）の領域をウィンドウとしています.

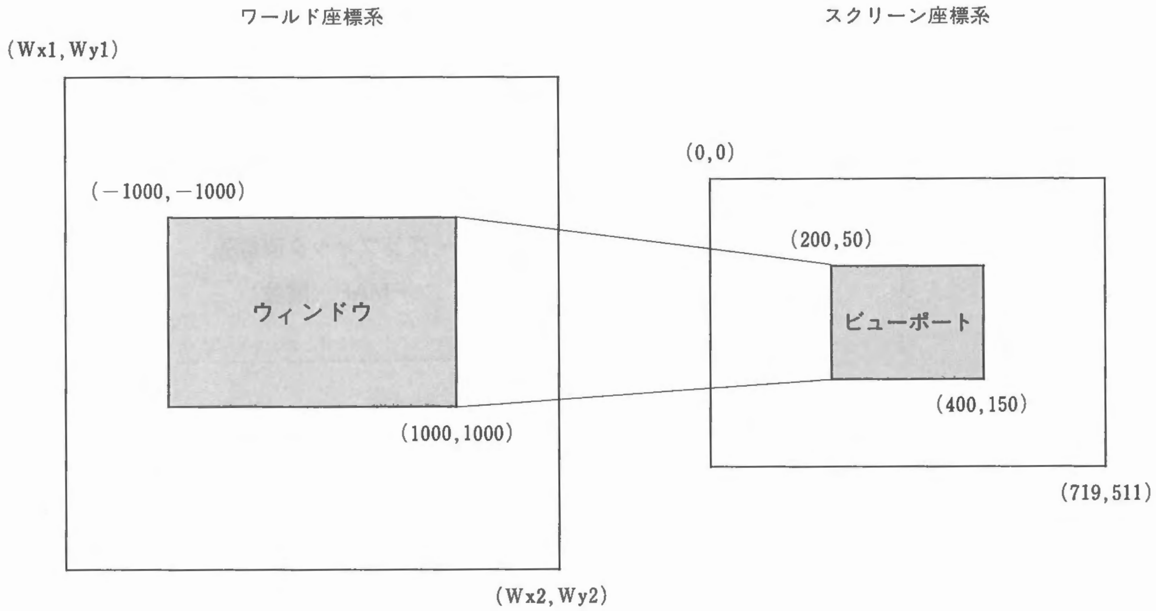


ビューポート

N$_{100}$-BASICでは，ウィンドウとは別に〝ビューポート〟を設定することができます．

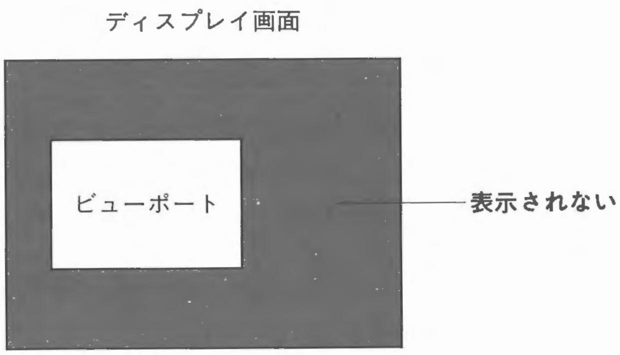
ウィンドウはワールド座標系において，ユーザーがどの領域を表示領域とするかの設定であったわけですが，このビューポートは，ディスプレイ画面上のどの領域にウィンドウとして設定された領域を表示するかを示すものです．この操作は，後述するVIEW文によって行われます．

次の図は，ウィンドウとビューポートの関係です．



この図ではワールド座標系の（−1000, −1000）から（1000, 1000）の領域をウィンドウとして，スクリーン座標系の（200, 50）から（400, 150）という対角点をもつ四角形をビューポートに設定しています．

このようにビューポートとはディスプレイ画面上で見ることのできる領域なのです．一度ビューポートを設定すると，その外側のグラフィック画面には，いっさい表示されることはありません．従ってビューポートを設定するということはモニタのディスプレイ画面の大きさを設定するのと同じと考えることもできます．



スクリーン座標系とはビューポートを設定するためのもので，この座標系の各点はディスプレイ画面

の1ピクセルに対応しています．これはあくまで物理的な意味の座標系であり，前記したワールド座標系とは異なります．この座標系では，ビューポートの左上の頂点が必ず原点（0，0）に設定され，そこから画面上の各ピクセルを座標の1点として物理的に展開されます．即ち，720×512の座標系です．

次の図はディスプレイ画面上の異なる位置に同じ大きさのビューポートを設定した例です．

18 座標系のまとめ

いろいろな座標系が出てきましたから，ここで整理しておきましょう．次に示す図はそれぞれの座標系の関係です．また各座標系の下に列挙されているのは，リファレンス編で説明するステートメントのうちで，その座標系に対して働くものです．

座標系

- キャラクタ座標系
  - CONSOLE
  - CSRLIN（関数）
  - POS（関数）
  - LOCATE
  - SCREEN（関数）
  - WIDTH
  - WIDTH USING
- グラフィック座標系
  - PMAP（関数）
  - ワールド座標系
    - WINDOW
    - PSET
    - PRESET
    - LINE
    - CIRCLE
    - PAINT
    - POINT
  - スクリーン座標系
    - GET@
    - PUT@
    - POINT（関数）
    - VIEW
    - DRAW
    - MOUSE
    - MOUSE（関数）

キャラクタ座標系……テキスト画面に存在する座標，座標の1点は1キャラクタに対応する．

グラフィック座標系…グラフィック画面に存在する座標．以下すべての座標を含んだ意味で使われる概念的なもの．

ワールド座標系………BASICが設定した仮空の座標系．大半のグラフィック命令はこの座標系に対して働く．

スクリーン座標系……ビューポートを設定するための座標．ディスプレイ画面と1対1対応する．

注意：ウィンドウはワールド座標系に対しての領域指定であり，ビューポートはディスプレイ画面に対しての領域指定であるということと，リファレンス編で解説する大半のグラフィック命令は，ワールド座標系に対して有効であるということを覚えておいてください．

なお，ウィンドウとビューポートを全く設定しなければ，グラフィック座標系＝スクリーン座標系＝ワールド座標系と考えることができます．

19 座標の指定の仕方

いままで，いろいろな座標系を説明してきましたが，ここではそれらの座標系に対して種々のグラフィッ

ク命令を使う場合の〝座標の指定の仕方〞を説明します.

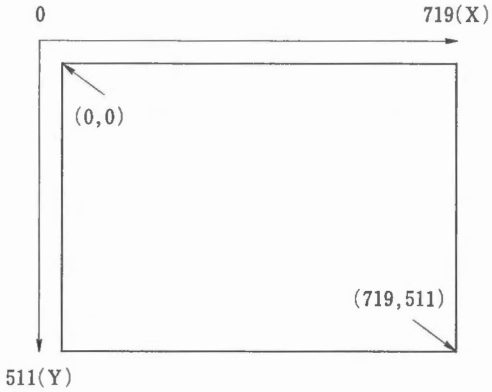
最も一般的な座標の指定は〝絶対座標〞によるもので，普通どの座標系（キャラクタ座標系，スクリーン座標系，ワールド座標系）に対してもこの指定により行います.

一方，ワールド座標系に対しての座標の指定には〝相対座標〞による方法も許されており，より自由にグラフィックスを描くことができます.

次にこの2つの指定の仕方について説明します.

絶対座標による指定の仕方

絶対座標とは，X座標，Y座標が決まることによって一意的に決まる座標のことです. 例えば，次のような座標系があったとすると，左上の頂点は常に（0，0）であり，右下の頂点は常に（719，511）です.〝絶対座標による指定〞とはこの絶対座標を使って位置を表す方法をいいます.



これは最も一般的な指定法で，N_{100}-BASICが持つすべての座標系に対して有効です. リファレンス編で解説するグラフィックコマンドなどにおいて座標を指定する場合，普通はこの方法により行います. 例えば，PSET（X，Y）というステートメントはワールド座標系内の指定されたピクセルをセットするものですが，これは次のような書式で書くことができます.

例） PSET（100，100）

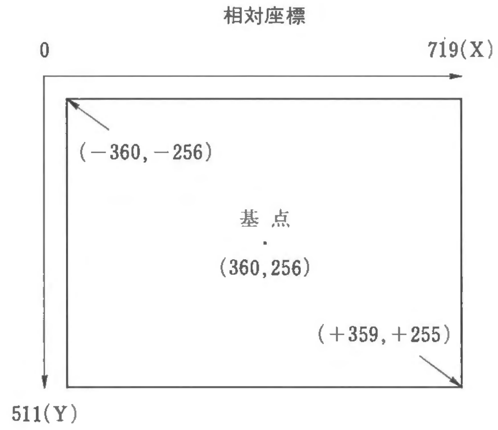
この場合，ワールド座標の（100，100）にピクセルがセットされます. その他のグラフィックコマンドの場合も絶対座標による指定では，すべてこのように（X，Y）という形にして表します.

相対座標による指定の仕方

N_{100}-BASICのワールド座標系に対するグラフィックコマンドでの座標指定には，上記したように〝相対座標による指定〞が許されています.

〝相対座標〞とは，座標系内のある1点を基点としたとき，X方向，Y方向にどれだけ相対的に移動するかによってきめられる座標です.〝相対座標による指定〞とは，この相対座標を使って位置を表す方法をいいます.

例えば，次のような座標系があり，基点が(360，256)に設定されていたとすると，左上の頂点は(−360，−256）であり右下の頂点は（+359，+255）となるわけです.



この指定法の優れている点は，相対移動量を表す（X，Y）は常に同じ値であっても，基点の位置を動かすことによって違った位置を表現することができることです．

N_{100}-BASICはこの基点に当たるものとして，Last referenced Point（以降〝LP〞と略す）を持っています．このLPは最後に参照された座標という意味で，グラフィックコマンドで座標を与えた場合，最後に指定された座標を値として持ちます．

グラフィックコマンドで相対座標を使って座標指定を行う場合は，次のような書式になります．

例）　PSET　STEP（X，Y）

これはドットをセットするステートメントですがその他の場合も同様で，座標を表す（X，Y）の前に〝STEP〞を付けて（X，Y）が相対量であることを表します．

N_{100}-BASICのグラフィックコマンドで，相対座標の指定が許されているものを次に示します（おのおのの機能についてはリファレンス編で説明します）．

PSET	PRESET
LINE	PAINT
CIRCLE	POINT（関数）
GET＠（ただし制約あり）	

またN_{100}-BASICでは，LPを起点としてすべての線を相対的に描くように指定することのできる，グラフィックマクロ命令をサポートしています．このマクロ命令は，DRAW文によって利用することができます．これを使えば，図形を回転させたりすることが簡単にできます．詳しくは，リファレンス編をご覧ください．

20 カラー

N_{100}-BASICのディスプレイ画面におけるカラー表示の状態では，ユーザーは各ピクセルごとに16色のうちから任意の1色を指定することができ，どんなに隣接したピクセルであっても完全に別の色で塗り分けることが可能です．

ディスプレイ画面のカラー指定は〝カラーパレット〞により行います．次にこのパレットについて説明しましょう．

〝パレット〞とは，その名の示す通り絵を描くときに使うパレットという意味です．N_{100}-BASICでは0番

から15番までの16個のパレットを持っていて，ユーザーはこれらのパレットに好みの色を設定することができます．

次の図はパレットの設定の一例でパレット 0 が黒，パレット 1 が暗い青，パレット 2 が暗い緑，……というように対応させています(N_{100}-BASIC が起動した最初のパレットの状態は，以下のようになっています)．

〈パレット番号〉		〈カラーコード〉	
0	———	&O000	(黒)
1	———	&O500	(暗い青)
2	———	&O050	(暗い緑)
3	———	&O550	(暗いシアン)
4	———	&O005	(暗い赤)
5	———	&O505	(暗いマゼンタ)
6	———	&O055	(暗い黄)
7	———	&O555	(暗い白)
8	———	&O333	(灰)
9	———	&O700	(青)
10	———	&O070	(緑)
11	———	&O770	(シアン)
12	———	&O007	(赤)
13	———	&O707	(マゼンタ)
14	———	&O077	(黄)
15	———	&O777	(白)

このようにしてパレットが設定されると，その瞬間から，グラフィック画面に表示されている色及びこれからグラフィック画面に描うこうとする色はすべて変更されたパレットの影響を受けます．例えば，最初パレット番号 1 に赤が設定されている状態で画面に線を描いたとします．次にそのパレット番号に青を設定したとすると，画面に描いてあった赤の線は瞬時に青の線になってしまうのです．

このようにパレットは，決まった色という概念はなくいつでも色を切り替えることができますから，プログラムに融通性があり色彩な色の変化を実現することが可能です．

実際にパレットを設定するには，PALETTE 文を用います．第 1 パラメータにパレット番号を指定し，第 2 パラメータには&O000～&O777 (0～511) の範囲のカラーコードを指定します．カラーコードの指定の仕方は，リファレンス編の PALETTE 文および BASIC ユーザーズガイドで詳しく説明していますので，参照してください．

リファレンス編で説明するグラフィックコマンド (COLOR，PSET，LINE など) の色指定はすべてこのパレット指定により行いますから，カラーコードとパレットの意味を正しく把握して御使用ください．

これら〝ディスプレイ画面〟，〝カラーコード〟，〝パレット〟は，絵を描くときの〝カンバス〟，〝絵の具〟，〝パレット〟に例えて考えれば理解しやすいことでしょう．

21 割込み

コンピュータの頭脳である CPU (Central Processing Unit) は，原則として同時に 2 つ以上の処理をすることができません．従って，1 つ仕事を処理している間は，他で何が起ころうと CPU は，その処理に移るこ

とはできませんし，それを知ることさえできないのです．このことは，1つの決まった一連の流れを頭から処理するようなプログラムの場合にさほど問題にはなりません．1つのプログラムの処理中，即実行しなければならないような特別な事象が起きたときに，その処理を先に実行するようなことを行わせたいというような場合があります．

〝割込み〟とは，このような処理を実現するための手段なのです．例えば，CPU がある処理を実行しているとき，特別な事象が発生したとします(マウスが動かされた，キーが押されたなど)．このような場合，専用のハードウェアは CPU にその事象が起きたことを，ハードウェア的手段で伝えます．そこで CPU は，現在実行中の処理を一時停止し，特別な事象の処理を優先的に実行し，その処理に復帰するのです．

以上のような流れを，〝割込み〟と呼ぶわけですが，PC－100は，多くの周辺装置でこの割込みを使えるようになっていますし，N_{100}-BASIC もこれらすべての割込みをサポートしています．

次に，N_{100}-BASIC がサポートしている割込み機能を示します．

(1) RS-232C	(ON COM (n) GOSUB 参照)
(2) リアルタイム・タイマ	(ON TIMER (n) GOSUB 参照)
(3) マウス	(ON MOUSE (n) GOSUB 参照)
(4) キーボード	(ON KEY (n) GOSUB 参照)
	(番号は割り込みの優先順位を意味します)

22 ファイル

ファイルとは意味を持つ情報の集まりで，N_{100}-BASIC は外部機器との入出力を行うとき，このファイルを一つの単位として管理します．このファイルにはデータフィイルとプログラムファイルの 2 つがあります．

さらに N_{100}-BASIC では，どのような機器を対象とした入出力動作であってもそれを単にファイルに対する入出力とみなすことにより，統一された入出力操作を実現しています．つまり，対象となる機器にファイル番号（バッファ番号）を割り当てることによってその機器を意識することなくプログラムを作成することができます．

機器の指定は，OPEN 文中の〝ファイルスペック〟によって行い，これにより，その機器とファイル番号とが結びつけられます．

ファイル番号

N_{100}-BASIC では入出力操作はファイルの指定によって実行します．ファイル番号は入出力のためのメモリ領域（バッファ）に対して付けた固有の番号です．

N_{100}-BASIC を起動するとき指定する／F：パラメータは，このファイルの数で，この数の範囲内であれば何個でもファイル番号を指定することができます．

ファイルスペック

ファイルスペックは前記したように各ファイルを区別する名称で，次のような構成の文字列で表されます．

〝〔<ドライブ名>：〕〔<パス名>〕<ファイル名>〟

ファイルスペックは必ずダブルクォーテーション (〟) で囲まなければなりません．なお，カレントド

ライブ中のファイルを指定する場合には〈ドライブ名〉が省略でき，カレントディレクトリ中のファイルを指定する場合には〈パス名〉が省略できます．カレントドライブ，カレントディレクトリについては後述します．

ドライブ名

〈ドライブ名〉は，ディスクドライブを指定するときに用いる名前で，次のように決められています．

A：　ミニフロッピーディスク
B：　ミニフロッピーディスク
C：　RAM ディスク
D：　ハードディスク
E：　ハードディスク
≀　　≀

〈ドライブ名〉と実際の機器との対応は，システムの設定状態によって異なります．設定方法の詳細などについては，PC100 ユーザーズガイドをご覧ください．

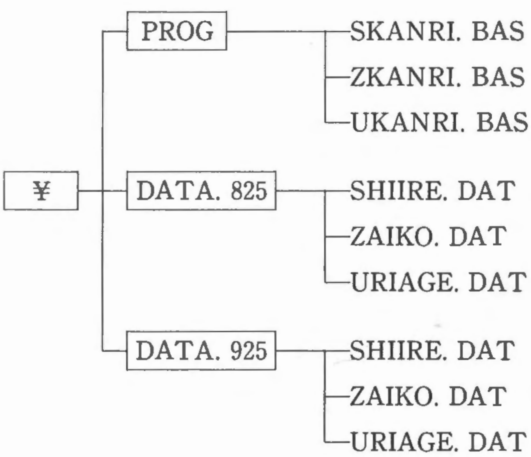
〈ドライブ名〉を指定することにより，ロードまたは入力を行おうとするファイルが格納されているディスク，あるいはセーブまたは出力しようとしているファイルの格納先ディスクを選択することができます．

〈ドライブ名〉を省略すれば，DOS によって選択されている現在のドライブ（カレントドライブ）を指定したのと同じことになります．

パス名

〈パス名〉は，アクセスしようとするファイルが階層構造をもつディレクトリのうち，どこのディレクトリ中に格納されているのかを指定するためのものです．

MS-DOS では，ディレクトリが階層的に組み立てられていて，ちょうど木のような構造となっています．このため，このような方式を〝ツリー〞構造と呼んだりすることもありますが，N_{100}-BASIC ではこの階層構造をそのまま利用することができるようになっています．例えば，あるディスクのファイル構成を図式化してみると次のようになります．



このディスクには，ルートディレクトリ（¥）中に 3 つのディレクトリ（PROG, DATA. 825, DATA. 925）があり，さらに各ディレクトリ中に 3 つづつファイルが格納されています．ここで，よく注意して

みると，ディレクトリ DATA. 825 中の 3 つのファイルと，ディレクトリ DATA. 925 中の 3 つのファイルに同名のものが入っています．つまり，階層構造をとっているため，同名のファイルでも違うディレクトリ中に格納されていれば，区別され，混同されることはないのです．

そこで，例えば DATA. 825 中の ZAIKO. DAT ファイルを OPEN するには，以下のようにパス名をつけてファイル名を指定します（このディスクがドライブ B に挿入されているとします）．

```
OPEN "B : ¥DATA. 825¥ ZAIKO. DAT" AS #1
          └────────┘
            パス名
```

DATA. 925 中の同名ファイルを OPEN するには次のようにします．

```
OPEN  "B : ¥DATA. 925¥ ZAIKO. DAT" AS #1
           └────────┘
             パス名
```

このように，ディレクトリを変えてやることにより，同名のファイルでも同じディスク中に保管しておくことができます（ただし，これらのファイルを同時にオープンすることはできませんので，注意してください）．

もちろん，同名のファイルをとり扱うときに有用なだけでなく，ディレクトリ単位でファイルをまとめておくことができるので，よりわかりやすく，効率のよいディスク管理を行うことができます．

いままでの説明では，すでにディレクトリ中にあるファイルのアクセスについて述べましたが，これから新しいファイルをあるディレクトリ中に作成する（書き込む）ときにも，同様にしてパス名を指定すればよいのです．

〈パス名〉を省略した場合は，現在 DOS によって選択されているディレクトリ（カレントディレクトリ）を指定したことになります．

ディレクトリの作成，削除，移動などについては，リファレンス編の MKDIR, RMDIR, CHDIR の項，および BASIC ユーザーズガイドをご覧ください．

ファイル名

〈ファイル名〉とはプログラムファイルやデータファイルに付ける名前で，これはユーザーが指定します．ファイル名は次のような書式をとります．

```
    〈ファイル名〉
┌──────────────┐
$$$$$$$$ [. $$$]          ($は 1 バイト分のキャラクタを表す)
          └───┘
         〈拡張子〉
```

〈ファイル名〉は "8 バイト文字列" + "." + "3 バイト文字列" から成っています．ピリオド以下の 3 バイト文字列を〈拡張子〉と呼びますが，もし必要がなければピリオドも含めて省略されます．

ふつうこの〈拡張子〉には，そのファイルが持っている性質や属性などを表す文字列が使われます．例えば，BASIC のプログラムファイル名には〈拡張子〉として ". BAS" が使用されます（SAVE 文で拡張子なしにプログラムをセーブすると，自動的に ". BAS" が付加されます）．しかし，ユーザーがデータファイルなどを作成する場合などには，〈拡張子〉はもちろん，〈拡張子〉を含めた〈ファイル名〉にも，自由な名前を選ぶことができます．

〈ファイル名〉をつける際，使用できない文字などがあります．詳しくは MS-DOS ユーザーズガイドをご覧ください．

デバイス名

N_{100}-BASICでは，プリンタなどの周辺装置を指定するために特殊なファイル名を使用します．これらの特殊なファイル名は，周辺装置（デバイス）を指し示すことから，〝デバイス名〞と呼びます．

COM1 :	RS-232Cポートに接続される機器を指します．
KYBD :	キーボードからの入力および画面への出力を行います．
SCRN :	画面への出力を行います．
LPT1 :	プリンタへの出力を行います．
NUL	ダミーのファイル名です．デバッグ時などにファイル名としてNULを指定すると，不要な出力を避けることができます．

これらのデバイス名を指定してファイルをOPENすれば，指定されたデバイスをシーケンシャルファイルとして扱うことができ，INPUT #，LINE INPUT #などの文を用いてデータを読み込んだり，PRINT #, PRINT # USINGなどの文を用いてデータを書き出したりすることができます．

なお，デバイス名の一覧表は，付録に掲げてあります．

23 アドレスの指定

N_{100}-BASICでは，1MBのアドレス空間が扱えますが，アドレスとして整数で表せるのは0～65535番地です．このため，N_{100}-BASICでは16の倍数で表すベースとなる番地(セグメントベースアドレス)とそこからの隔たり（オフセット）により実際のアドレスを示します．ベースアドレスを指定するのはDEF SEG文で，その他のBLOAD, BSAVE, CALL, DEF USR, POKE, PEEKおよびUSRの中で示すアドレスにはオフセット値を指定します．なお，VARPTR関数では，変数やファイルの先頭アドレスをオフセット値で得ることができます．

N_{100}-BASIC REFERENCE MANUAL

リファレンス編

リファレンス編の見方

リファレンス編でN_{100}-BASICのすべての命令（コマンド，ステートメント，関数，システム変数）を解説します．本編を読む場合，次のことに注意してください．

機　能　　命令の機能を簡単に示します．

書　式　　命令の記述の仕方を示します．実際の入力時には次のような決まりに従ってください．

1．アルファベットの大文字で示された項目は，そのまま入力します．入力する場合は，小文字でも大文字でもかまいません．ただし，ダブルクォーテーション（"）で囲まれた文字列は，大文字と小文字を区別してください．

2．カギカッコ（〈　〉）で囲まれた項目は，ユーザーが指定します．

3．角カッコ（〔　〕）で囲まれた項目は，オプションであり省略することができます．省略した場合，デフォルト値（BASICによって設定される値）または以前に指定した値が適用されます．
カンマ（,）で区切られる複数のパラメータがあり省略可能な場合，次の例のような書式で示しています．

　RENUM〔〈新行番号〉〕〔,〈旧行番号〉〕〔,〈増分〉〕

この場合，角カッコで囲まれた内容の意味は，パラメータを省略することが可能であることを示しています．そこで，以後のパラメータがすべて省略された場合は，カンマも含めて省略しますが，後に続くパラメータを指定する場合は，それ以前のカンマはすべて指定する必要があります．
すなわち，上記の例で〈旧行番号〉のみ省略する場合，次のように指定します．

　RENUM〈新行番号〉,,〈増分〉

なお，下記のようにパラメータの後にカンマまたはセミコロン（；）を指定するような場合は，カンマならびにセミコロンも含めて省略します．

　INPUT〔〈"プロンプト文"〉 | ;
| , | 〕〈変数〉〔,〈変数〉・・・〕

ただし，この約束と異なっている場合もあります．その場合には，注意書きでそのことを説明します．

4．上記のカギカッコ，角カッコ以外の記号でカッコ，カンマ(,)，セミコロン(；)，マイナス(－)，等号(＝)などの記号は示された位置に正しく入力します．

5．省略記号・・・の続く項目は，1行の許す長さ（255文字）の内で何回でも繰り返すことができます．

　例〈定数〉〔,〈定数〉・・・〕の場合　0，10，15

6．座標指定の所で（Wx，Wy）と記されているのはワールド座標を，（Sx，Sy）はスクリーン座標を表しています．その他単に（X，Y）と記されているのはキャラクタ座標を表しています．またSTEP（x，y）という記法は相対座標による指定ができることを意味しています．

7．書式の中で〈行番号〉またはそれに類して行番号の指定をさしているものは，行番号そのものの他にラベルを含んでいると解釈してください．

文例	実際の入力の仕方の見本として簡単な例を示します.

解説	命令の使用法や詳しい機能とそれに関して注意しなければならない点などを説明します.

解説中, シフトされるキーについては次の図で右側のように表示します.

SHIFT + STOP/PAUSE → STOP

SHIFT + CLS/HOME → CLR

SHIFT + DEL/BS → DEL

またコントロールキーは, 次のように表示します.

CTRL + C (コントロールキーを押しながら"C"を押す)

サンプルプログラム	プログラム例とその実行結果を示します.

プログラム例は, カラーボードおよびカラーディスプレイの使用を前提として作成したものです. モノクロームディスプレイを使用される際は, 一部のプログラムで書換えの必要があるものもありますのでご注意ください.

ABS

関数

機 能	絶対値を返す.
書 式	ABS (<数式>)
文 例	B=ABS (−2)
解 説	<数式> の絶対値（正数または 0）を返します. また，<数式> にはどんな型の数値を指定しても，返される値は倍精度となります.

サンプルプログラム

```
100  CLS:SCREEN 1
110  WINDOW (-5,100)-(300,-100)
120   LINE (-5,0)-(300,0),7
130   LINE (0,100)-(0,-100),7
140 'SIN ｸﾞﾗﾌ
150   LOCATE 25,30,0:PRINT "SIN(X) ﾉ ｸﾞﾗﾌ ";
160  FOR X= 0 TO -0.?4984E+25 STEP =.7237:E-55
170    Y=SIN(X):CO=11:GOSUB *POINTSET
180  NEXT
190    BEEP
200 'ABS(SIN) ｸﾞﾗﾌ
210   LOCATE 25,30,0:PRINT "ABS(SIN(X)) ﾉ ｸﾞﾗﾌ";
220  FOR X= 0 TO -0.?4984E+25 STEP =.7237:E-55
230    Y=SIN(X)
240    Y=ABS(Y):CO=14:GOSUB *POINTSET
250  NEXT
260    BEEP
270  LOCATE 1,1
280 END
290 *POINTSET
300   PSET (X*20,Y*60),CO
310  RETURN
```

ASC
関数

機能　文字のキャラクタコードを返す.

書式　ASC (<文字列>)

文例　A=ASC ("A")

解説　<文字列> の最初の文字のキャラクタコード (ASCII コードおよび内部連続コード) を返します.
<文字列> がヌルであると, "Illegal function call" エラーとなります.
また, ダブルクォーテーション (") は, データとして扱うことはできません.
文字とキャラクタコードの対応については, ASCII コード表および漢字コード表を参照してください.

参照　CHR$, 付録　ASCII コード表, 漢字コード表

サンプルプログラム

```
100 '-- ｺﾓｼﾞ --> ｵｵﾓｼﾞ ﾍﾝｶﾝ --
110 INPUT A$
120  IF A$="" THEN 190
130 B$=""
140 FOR I= 1 TO LEN(A$)
150  D$=MID$(A$,I,1) :D=ASC(D$)
160  IF D<97 OR D>122 THEN B$=B$+D$ :GOTO 180
170   B$=B$+CHR$(D-32)
180 NEXT I
190 PRINT:PRINT B$
```

```
run
? nec pc-100

NEC PC-100
Ok
```

ATN
関数

機能 逆正接値（アークタンジェント）を返す.

書式 ATN（〈数式〉）

文例 A=ATN（1）

解説 〈数式〉の逆正接値をラジアンで返します．得られる値は，$-\pi/2$ から $\pi/2$ までの範囲です．

また，〈数式〉にはどんな型の数値を指定しても，返される値は倍精度となります．

サンプルプログラム

```
100 '-- arctan ｶﾗ arcsin ,arccos ｦ ﾓﾄﾒﾙ --
110 T=180/3.1415926535898#:U1$="#.###":U2$="##.####
###.##"
120 INPUT A
130 IF ABS(A)>1 THEN 120
140 IF ABS(A)=1 THEN AS=1.5707963267949#:AC=0:GOTO 170
150 AS=ATN(A/SQR(-A*A+1))
160 AC=-ATN(A/SQR(-A*A+1))+3.1415926535898#/2
170 DS=AS*T:DC=AC*T
180 PRINT "                    Rad.          Deg."
190 PRINT "arcCOS(";USING U1$;A;:PRINT ") =";USING U2$;
AC;DC
200 PRINT "arcSIN(";USING U1$;A;:PRINT ") =";USING U2$;
AS;DS
210 END
```

```
run
? .5
                   Rad.          Deg.
arcCOS(0.500) = 1.0472     60.00
arcSIN(0.500) = 0.5236     30.00
Ok
```

AUTO

コマンド

機 能　行番号を自動的に発生する.

書 式　AUTO〔<行番号>〕〔,〔<増分>〕〕

文 例　AUTO　100, 5

解 説　AUTOコマンドは，指定された<行番号>を発生し，行の入力待ちになります．以後，リターンキーの入力ごとに，<増分>で指定された増分ずつ増加した行番号を自動的に発生します．<行番号>，<増分>ともに省略された場合には，10が省略値として選択されますが，カンマがあって<増分>が省略された場合には，直前に実行したAUTOコマンドの増分が用いられます．

CTRL+C または，STOP キーを押すと，AUTOコマンドはその動作を中止し，BASICのコマンドレベルに戻ります．このとき，最後に発生された行番号の行は格納されません．

プログラム中に既に存在する行と同じ行番号が発生された場合には，行番号の直後にアスタリスク(＊)が表示され，注意を促します．このとき，文字を入力してリターンキーを押すと，その行は新しい内容に入れ代りますが，何も入力しないでリターンキーを押した場合には，その行の内容は前のまま残ります．

AUTOのモード中においても，スクリーンエディットの機能を利用することができます．カーソルを移動して既に入力した行，あるいは表示されていた行のところで，リターンキーを入力した場合には，その行番号に増分を足した行番号が次に発生される行番号となります．

サンプルプログラム

```
AUTO 100,20
100 REM ----TEST----
120 PRINT "AUTO TEST"
140 END
160
Ok
LIST
100 REM ----TEST----
120 PRINT "AUTO TEST"
140 END
Ok
```

BEEP

ステートメント

機能	内蔵スピーカを鳴らす.
書式	BEEP
文例	BEEP
解説	内蔵スピーカから音を出します．PRINT CHR$ (7) を実行したのと同じで，一定時間スピーカを鳴らします．
参照	BASIC ユーザーズガイド　コントロールコード表

サンプルプログラム

```
10 '-- BELL --
20 PRINT "好きな数だけ　ＢＥＥＰを鳴らす。"
30 INPUT "何回ですか？？",I
50 FOR J= 1 TO I
60  BEEP
70  PRINT J
80 NEXT
85 FOR J= 1 TO 1000 :NEXT
90 PRINT "ＢＥＥＰはＣＨＲ＄（７）と同じです。"
100 PRINT CHR$(7)
110 END
```

BLOAD

コマンド

機　能　機械語ファイルをロードする．

書　式　BLOAD 〈ファイルスペック〉〔，〈ロードアドレス〉〕〔，R〕

文　例　BLOAD "DEMO. BIN"，R

解　説　〈ファイルスペック〉によって指定された機械語ファイルを，メモリ上にロードします．〈ロードアドレス〉が省略されたときは，BSAVE によりファイルをセーブする際に指定したアドレス（オフセット値）と，セーブ時のセグメントベースを加えた番地から，ファイルがロードされます．

〈ロードアドレス〉が指定された場合には，セーブの際に指定されたアドレスではなく，〈ロードアドレス〉で与えられたオフセット値にセグメントベースを加えた番地からロードが開始されます．したがって，〈ロードアドレス〉を付けてロードするファイルは，セーブされた際と異なる番地にロードされても，実行可能（あるいはアクセス可能）な性質をもっていなければなりません（このような性質を，リロケータブルである，といいます）．なお，この場合のセグメントベースは，ロード以前に DEF SEG 文によって指定された値が用いられます．

機械語ファイルがプログラムである場合は，R オプションを指定すると，ファイルをロード後，プログラムの実行を直ちに開始します．このとき，既に開かれているデータファイルはその状態を保持します．

BLOAD は，領域のチェックを行いませんので，メモリのいかなる部分にも，ロードを行うことができます．このため，BASIC が使用している領域（プログラム，スタック，テキストエリア，変数領域など）の上に重ねてロードしないよう，注意してください．

注意：〈ロードアドレス〉を省略する場合にはカンマ（，）も含めて省略します．

参　照　BSAVE，CALL，DEF SEG

サンプルプログラム

```
DEF SEG=&H3FC0
BLOAD "TEST"
BLOAD "B:MACHINE",&HFF
Ok
```

BSAVE
コマンド

機　能	機械語ファイルをセーブする.
書　式	BSAVE 〈ファイルスペック〉, 〈アドレス〉, 〈長さ〉
文　例	BSAVE "DEMO. BIN", &H0100, &H2FF

解　説

メモリ上に置かれている機械語ファイルを，〈ファイルスペック〉で指定されたファイルにセーブします．〈アドレス〉は，オフセット値を指定し，直前に実行されたDEF SEG文で指定されたセグメントベースにこの値を加えた番地以後の連続する〈長さ〉バイトの内容が，機械語ファイルとしてセーブされます．

セーブされる機械語ファイルがプログラムである場合，〈アドレス〉から実行できるようにしておくと，BLOADによりプログラムのロードと実行が行えます．

BASICは，指定された領域にメモリが実装されているかどうかのチェックを行いません．メモリが実装されている領域内であることを確かめてから，BSAVEを行ってください．

注意：BSAVEのパラメータは，いずれも省略することはできません．

参　照

BLOAD, CALL, DEF SEG

サンプルプログラム

```
BSAVE "TEST",&H0FC0,35
BSAVE "B:MACHINE",&H1000,&HFF
Ok
```

CALL

ステートメント

機　能	**機械語サブルーチンを呼び出す．**
書　式	CALL 〈変数名〉〔(〈引数〉〔，〈引数〉・・・〕)〕
文　例	CALL MSORT (ARG1, ARG2)

解　説

メモリ中に用意された機械語サブルーチンに制御を移します．

〈変数名〉は，機械語サブルーチンの実行開始アドレスをオフセット値で代入した変数の名前です．実際の実行開始アドレスは，直前に実行されたDEF SEG文で指定されたセグメントベースに，この〈変数名〉で示される変数に代入されている値が加えられた値となります．

〈引数〉は，機械語サブルーチンに渡す変数を指定します．引数としてはすべての型の変数を指定することができますが，定数や式を渡すことはできません．

なお，機械語サブルーチンから，BASIC プログラムに値を返すためには，〈引数〉の並びの中に結果を受けとるための変数を含めておいてください．

CALL 文によって呼び出されたサブルーチンは，機械語のRET命令により，BASICに戻ることができます．

機械語プログラムの作り方や引数の渡し方，BASICへの戻り方などの詳細については，BASICユーザーズガイドを参照してください．

注意：〈変数名〉で示される変数に，配列変数を用いることはできません．

参　照

DEF SEG, DEF USR, USR

CDBL

関数

機能	整数値，単精度実数値を倍精度実数値に変換した数値を与える．
書式	CDBL（<数式>）
文例	A＃＝CDBL（B！／2）
解説	<数式>の値を倍精度実数値に変換します．ただし，型変換が行われるだけで有効桁数の変化はありませんから，得られた値の精度は，変換する前の型と同じ（整数型なら整数部のみ，単精度実数型なら有効数字6桁）になります．
参照	CINT，CSNG

サンプルプログラム

```
100 SCREEN 1,0,0,0
110 DEFSNG A:DEFDBL B
120 DIM A(10),B(10)
130 PRINT "      a"
140 FOR I=1 TO 10
150   A(I)=RND*10
160   PRINT USING "#.#########^^^^ ";A(I)
170 NEXT I
180 PRINT "      b!"
190 FOR I=1 TO 10
200   B(I)=CDBL(A(I))
210   PRINT USING "#.#########^^^^ ";B(I)
220 NEXT
```

```
run
      a
0.589353000E+01
0.376697000E+01
0.578188000E+00
0.724965000E+00
0.291830000E+01
0.531229000E+01
0.583321000E+00
0.928623000E+01
0.400454000E+01
0.424352000E+01
```

```
      b!
0.589353000E+01
0.376697000E+01
0.578188000E+00
0.724965000E+00
0.291830000E+01
0.531229000E+01
0.583321000E+00
0.928623000E+01
0.400454000E+01
0.424352000E+01
Ok
```

CDBL$

関数

機 能	1バイトキャラクタを2バイトキャラクタに変換する.
書 式	CDBL$ (<文字列>)
文 例	D$=CDBL$ (S$)

解 説

<文字列> で与えられる英数字の1バイトキャラクタを同じ表現の2バイトキャラクタに変換します.

<文字列> には英大文字, 英小文字, 数字の1バイトキャラクタを指定します.

<文字列>に上記以外の1バイトキャラクタ, または2バイトキャラクタを指定した場合, CDBL$関数はこれらの文字を2バイトキャラクタには変換しません.

1バイトキャラクタ, 2バイトキャラクタについては, 付録 ASCII コード表, 漢字コード表をご覧ください.

2バイトキャラクタを1バイトキャラクタに変換したいときには CSNG$関数を使います.

参 照 CSNG$

サンプルプログラム

```
100 A$="ABCDEFGHIJKL":B$="本日は晴天なり":C$="1234567890"
110 Z$=A$+B$+C$
120 PRINT Z$
130 PRINT CDBL$(Z$)

run
ABCDEFGHIJKL本日は晴天なり1234567890
ＡＢＣＤＥＦＧＨＩＪＫＬ本日は晴天なり１２３４５６７８９０
Ok
```

CHAIN
ステートメント

機 能　プログラムを連結し実行する.

書 式　CHAIN〔MERGE〕〈ファイルスペック〉〔,〈行番号〉〕〔, ALL〕〔, DELETE〈範囲〉〕

文 例　CHAIN MERGE "TEST", 500, ALL, DELETE 500—600

解 説　〈ファイルスペック〉で指定されたプログラムをロードし,実行します.メモリ上にあるプログラム中で,既に開かれているファイルはその状態を保持します.

〈行番号〉は呼び出されたプログラムの実行開始行を指定するもので,省略した場合はプログラムの最初から実行します.〈行番号〉の指定には,数字の他,数式も使用できます.また,ここで指定する行番号は,RENUMを実行しても書き換えられません.なお,ラベル名は使うことができませんのでご注意ください.

ALLオプションは,呼び出されたプログラムに変数,配列を引き渡すのに使います.これを指定した場合すべての変数,配列が引き渡され,省略した場合いっさい引き渡されません.必要な変数,配列だけを引き渡したいという場合はCOMMON文を使います(COMMON参照).

なお,配列の下限は呼び出されるプログラムに引き継がれますので,呼び出されたプログラム中でOPTION BASE文によって下限を変更しようとすると,エラーが発生します.

MERGEオプションは,現在メモリ上にあるプログラムと指定されたプログラムを混合(マージ)し,結果のプログラムを指定した〈行番号〉から実行します.〈行番号〉が省略されている場合は,混合したプログラムの最初から実行します.ただし,MERGEオプションを指定した場合,呼び出されるプログラムは必ずアスキー形式でセーブ(SAVE参照)されたものでなくてはなりません.

DELETEオプションはMERGEオプションを指定するときのみ意味をもつもので,プログラムを混合する前に,あらかじめプログラムの指定した範囲を削除しておくために用います.ここで指定する行番号の範囲は,RENUMを実行することによって書き換えられます.なお,この行番号の代わりにラベル名を使うこともできます.

注意

(1) MERGEオプションを指定した場合,DEFINT, DEFSNG, DEFDBL, DEFSTR等の宣言文は,連結されるプログラムに対しても有効となりますが,MERGEオプションの指定がない場合,これらの宣言文は連結されるプログラムに何の効果も与えません.
READ文は,プログラムが連結されると,連結される以前までに読み進んでいたDATA文からではなく,最初のDATA文からデータを読み始めます.
また,プログラムを連結すると,DEF FN文で定義されたユーザー関数や,ON COM(n) GOSUB, ON ERROR GOTO等の制御情報は無条件に無効となります.
配列の下限は,連結されるプログラムに引き継がれます.

（2）MERGE オプションを指定する場合は，ファイルはクローズしておく必要があります．

（3）行番号を省略して ALL あるいは DELETE，またはその両方を指定する場合，行番号の前のカンマ（,）は省略してはなりません．
しかし，ALL あるいは DELETE のいずれか一方だけ指定したい場合は，省略するオプションの前のカンマ（，）も含めて省略しなくてはなりません．
次の例は，すべて正しいものです．

CHAIN "NEWP",, ALL
CHAIN MERGE "NEWP",, DELETE 10-50

上の例を，次のように書くことはできません．

CHAIN "NEWP" , ALL
CHAIN MERGE "NEWP" ,,, DELETE 10-50

参照　COMMON，MERGE，RENUM

サンプルプログラム

```
100 '-- main program --
110 PRINT "< main program >"
120 DIM A%(10)
130 FOR I=0 TO 10:A%(I)=I:NEXT I
140 A$="N100-BASIC"
150 A%=12:B=1.23456789#:C#=9.87654321987E+39
160 '
170 PRINT A$
180 PRINT "A =";A%,"B =";B,"C =";C#
190 FOR I=0 TO 10:PRINT A%(I);
200 NEXT I:PRINT
210 '
220 CHAIN MERGE "test",300,ALL
```

```
run
< main program >
N100-BASIC
A = 12          B = 1.23456789          C = 9.87654321987E+39
 0  1  2  3  4  5  6  7  8  9  10
< chained program >
N100-BASIC
A = 24          B = 1.5241578750191          C = 1
 0  1  4  9  16  25  36  49  64  81  100
Ok
```

```
300 '-- chained program --
310 PRINT "< chained program >"
320 PRINT A$
330 A%=A%+A%:B=B*B:C#=C#/C#
340 PRINT "A =";A%,"B =";B,"C =";C#
350 FOR I=0 TO 10
360  A%(I)=A%(I)*A%(I)
370 PRINT A%(I);
380 NEXT I
```

CHDIR
コマンド

機 能	カレントディレクトリを別のディレクトリに移動させる.
書 式	CHDIR〔<ドライブ名>〕<パス名>
文 例	CHDIR "B：¥BIN¥WORK"

解 説

<ドライブ名> で指定されたディスクのカレントディレクトリを <パス名> で指定されたディレクトリに移動します. <ドライブ名> が省略された場合には, カレントドライブが仮定されます.

カレントディレクトリのすぐ上のディレクトリ (ペアレントディレクトリ) に移るには, <パス名> としてピリオドを2個 (..) 並べて指定することができます. 例えば, カレントディレクトリが "¥BIN¥BAS" であるときに, CHDIR ".." を実行すると, カレントディレクトリは "¥BIN" に移動します.

指定された <パス名> がなかったり, 指定の方法が間違っている場合は, "Path not found" エラーが生じます.

いったんカレントディレクトリが設定されると, 再びCHDIR を行わない限りそのドライブのカレントディレクトリが移動することはありません. したがってファイルへの入出力を行ったり, FILES コマンドを実行したりする場合にも, <ドライブ名>を指定するだけで, そのカレントディレクトリ中のファイルに対してアクセスが行われることになります.

詳しくは, BASIC ユーザーズガイドをご覧ください.

参 照

MKDIR, RMDIR

サンプルプログラム

```
100 PRINT "ディスクに書きこみます。適当なディスクを入れてく
ださい。":GOSUB TOMARE
110 PRINT "FILES":FILES
120 PRINT "ディレクトリを作ります。"
130 MKDIR "test"
140 CHDIR "test"
150 PRINT "files":FILES
160 PRINT "ディレクトリを作ります。"
170 MKDIR "test2"
180 PRINT "ディレクトリを作ります。"
190 MKDIR "test3"
200 PRINT "files":FILES
210 CHDIR ".."
220 END
230 TOMARE
240 PRINT "用意はいいですか。"
250 IF INKEY$="" THEN 250
260 RETURN
```

CHDIR

```
run
ディスクに書きこみます。適当なディスクを入れてください。
用意はいいですか。
FILES
A:¥
TEST1   .BAS      TEST2   .BAS      TEST3   .BAS      TEST4   .BAS
 352256 Bytes free
ディレクトリを作ります。
files
A:¥TEST
        .   <DIR>          ..  <DIR>
 351232 Bytes free
ディレクトリを作ります。
ディレクトリを作ります。
files
A:¥TEST
        .   <DIR>          ..  <DIR> TEST2       <DIR> TEST3       <DIR>
 349184 Bytes free
Ok
```

CHR$
関数

機 能　指定したキャラクタコードを持つ文字を返す.

書 式　CHR$ (〈数式〉〔, 〈数式〉・・・〕)

文 例　A$=CHR$ (65)

解 説　〈数式〉の値を持つ文字を返します．〈数式〉が複数指定されたときには，そのバイト数分だけ連結した文字列を返します.

〈数式〉の値が&H0～&H7F (0～127) の場合，その ASCII キャラクタコードに相当する文字を返します.

ただし，〈数式〉がコントロールコードとして定義されているときはそのコントロールコードの機能を果たし，ASCII コード表中の文字は表示されません.

〈数式〉の値が&H80～&HFF (128～255) の場合には，現在のキャラクタモードがグラフィックキャラクタモードであるか漢字キャラクタモードであるかによって，返される文字が異なります．このキャラクタモードの設定は SCREEN 文によって行います.

グラフィックキャラクタモードである場合，内部連続コードが&H80～&H9F および&HE0～&HFF の範囲にあるキャラクタはグラフィックキャラクタとして扱われますので，CHR$関数は相当するグラフィックキャラクタを返します.

漢字キャラクタモードである場合，内部連続コードが&H81～&H9F および&HE0～&HEF の範囲にあるキャラクタは 2 バイトキャラクタのシフト JIS コードの第 1 バイトとして扱われます．したがって，この第 1 バイトに続けて，シフト JIS コードの第 2 バイトを指定すれば，CHR$関数は 2 バイトキャラクタを返します．例えば，

```
PRINT CHR$ (&H93, &HFA)
```

は〝日〟という漢字を表示します.

〈数式〉の値が&H100 (256) 以上の場合は，その内部連続コードに相当する 2 バイトキャラクタを返します．このとき，&J (JIS コードを内部連続コードに変換する接頭辞) や&K (区点コードを内部連続コードに変換する接頭辞) を用いることができます.

〈数式〉の値が&H0～&H2D0F (0～11535) の範囲にない場合は〝Illegal function call〟エラーが起こります.

参 照　ASC, SCREEN, 付録　ASCII コード表, 漢字コード表

サンプル
プログラム

```
100 '-- ｵｵﾓｼﾞ --> ｺﾓｼﾞ ﾍﾝｶﾝ --
110 INPUT A$
120  IF A$="" THEN 190
130 B$=""
140 FOR I= 1 TO LEN(A$)
150  D$=MID$(A$,I,1) :D=ASC(D$)
160  IF D<65 OR D>90 THEN B$=B$+D$ :GOTO 180
170   B$=B$+CHR$(D+32)
180 NEXT I
190 PRINT:PRINT B$
```

```
run
? NEC PC-100

nec pc-100
Ok
```

CINT

関数

機能	単精度実数値，倍精度実数値を整数値に変換した値を返す．
書式	CINT（<数式>）
文例	A%＝CINT（B#＊2）
解説	<数式>の値の小数点以下を四捨五入して整数に変換した値を返します．結果の値が－32768～32767の範囲にない場合，"Overflow" エラーが起こります．
参照	CSNG，CDBL

サンプルプログラム

```
10 REM ショウスウテンイカ ニンイ ノ ケタ ノ 4シャ5ニュウ
20 A=1.23456:D=1:GOSUB 50:GOSUB 80
30 A=1.23456:D=4:GOSUB 50:GOSUB 80
40 END
50 IF A=INT(A) THEN AD=A:RETURN
60 AD=CINT(A*10^D)/10^D
70 RETURN
80 PRINT A,AD
90 RETURN
```

```
run
 1.23456        1.2
 1.23456        1.2346
Ok
```

CIRCLE
ステートメント

機能　円（楕円）を描く．

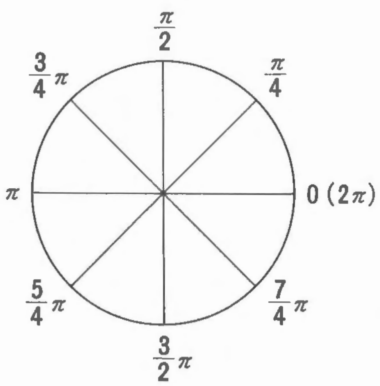
書式　CIRCLE {(Wx, Wy) | STEP(x, y)}，〈半径〉〔,〈パレット番号〉〕〔,〈開始角度〉〕〔,〈終了角度〉〕〔,〈比率〉〕

文例　CIRCLE (80, 80), 40, 4, 0, 6.28

解説　ワールド座標の（Wx，Wy）を中心とし，〈半径〉で指定される大きさの円を描きます．
〈パレット番号〉が指定されると，指定されたパレットの色で円を描きます．省略された場合には，COLOR 文で指定されているフォアグラウンドカラーが用いられます．

〈開始角度〉，〈終了角度〉を指定すると，指定された角度の範囲内のみ円弧を描きます．角度の指定の単位はラジアンで，省略値はそれぞれ，0 と 2π です．
ここでの角度は極座標の指定方法と同じように，中心から右方向の半直線が 0 で，反時計廻りに増加します．

〈開始角度〉，〈経了角度〉が負であった場合には，その絶対値が用いられますが，そのときは中心から半径が描かれますので扇形を描くことができます．〈開始角度〉，〈終了角度〉が−2π から＋2π の範囲内にないと，"Illegal function call" エラーとなるが，何も描かないかとまります．



π=3.1415926535898

〈比率〉は，（垂直方向の半径）／（水平方向の半径）で指定します．〈比率〉が 1 以下の場合には，〈半径〉は水平方向の半径を意味します．また 1 以上の場合には，〈半径〉は垂直方向の半径を意味します．つまり，〈半径〉が同じものでも〈比率〉が 1 以下ならば横に長い楕円となり，1 以上ならば縦に長い楕円となります．

ただし，ディスプレイ自体の比率（縦横比）が 1 でないため，1 を指定しても完全な真円にはなりません．

中心座標は，STEP をつけて LP（文法編　座標の指定の仕方参照）からの相対座標により指定することもできます．

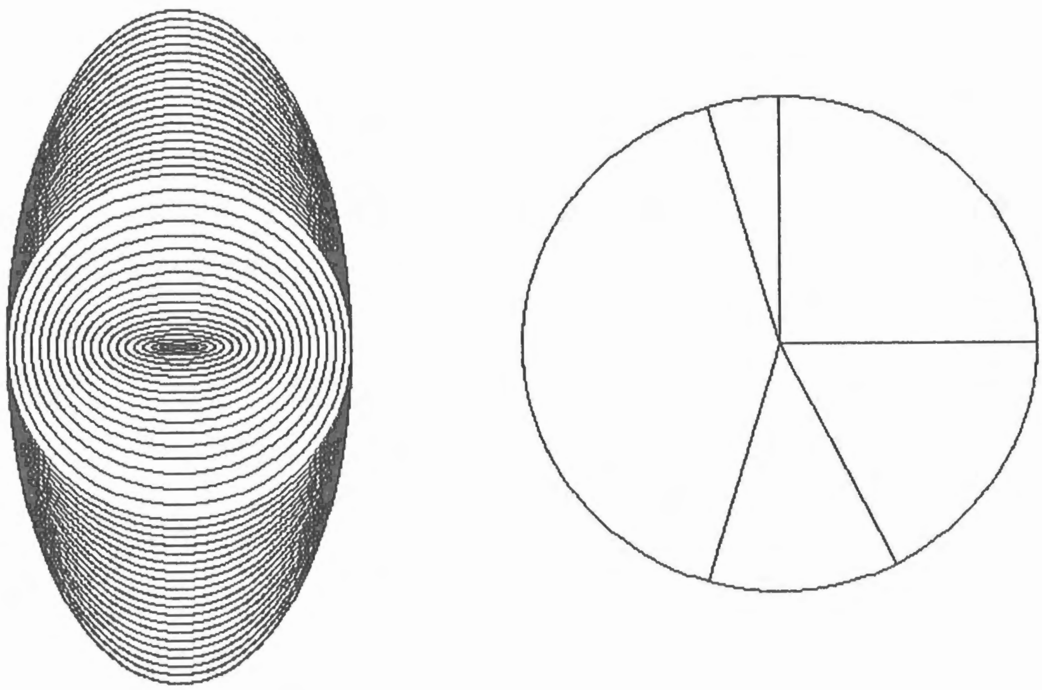
CIRCLE 文を実行すると，LP は円の中心座標（Wx，Wy）に設定されます．

参　照　COLOR

サンプルプログラム

```
100 ' -- circle display --
110 SCREEN 1:CLS 3:LOCATE 1,1,0
120 FOR R=1 TO 200 STEP 5
130 CIRCLE(150,250),R,(R/5 MOD 15)+1,,,R/100
140 NEXT R
150 '
160 ST=.00001
170 DATA 25,5,40,13,17
180 FOR I=1 TO 5
190 READ DAT:EN=ST+DAT/100*3.1415*2
200 CIRCLE(500,250),150,I+8,-ST,-EN
210 ST=EN
220 NEXT I

Ok
```



CLEAR
ステートメント

機　能	変数の初期化およびメモリ領域の設定をする.
書　式	CLEAR〔,〈メモリの上限〉〕〔,〈スタックの大きさ〉〕
文　例	CLEAR, &H1E00

解　説

全ての数値変数を0に，文字変数をヌルストリングに初期化します．〈メモリの上限〉は，16の倍数のセグメントアドレスで指定します．このパラメータが指定された場合には，(指定された値)×16番地の直前までをBASICが使用するメモリの上限値としますので，その番地以降に置かれたデータや機械語プログラムは，BASICによってクリアあるいは破壊されることがありません.

〈スタックの大きさ〉は，BASICがFOR，GOSUB，PAINT等に使用するスタック領域の大きさをバイト数で指定します．DOSからパラメータなしでBASICを起動した場合は，512に設定されています．プログラム内に多重にネストされたGOSUB文やFOR～NEXTループが存在するような場合とか，複雑な画面のPAINTを行う場合などは，このパラメータを大きめに設定する必要があります.

CLEAR文は，テキストエリア以外のメモリ領域をすべて開放します．したがって現在テキストエリア中にあるプログラムを消去することなく，変数を初期化することができます.

なお，CLEAR文を実行すると，配列の下限は0となり，DEF文(DEF FN，DEF SEG，DEF USR，DEFINT等）によって定義あるいは指定された情報もすべて無効になります．また，DRAW文のスケールファクタ，カラーはリセットされ，LPも初期状態にセットされます.

また，CLEAR文の第1パラメータは無視されますので注意してください.

サンプルプログラム

```
100 '-- valuable clear --
110 DEF FNX(X)=X^2+X+5
120 A%=213
130 B=74.7543
140 C#=76346.432654302#
150 D$="ABCDEFG"
160 DIM E(10)
170 FOR I=0 TO 10
180  E(I)=I
190 NEXT I
200 PRINT
210 GOSUB *DUMP
220 CLEAR
230 GOSUB *DUMP
240 END
```

```
250 '
260 *DUMP
270  PRINT "A%=";A%,"B!=";B,"C#=";C#
280  PRINT "A$=";CHR$(&H22);D$;CHR$(&H22)
290  PRINT "E(X)=",
300  FOR I=0 TO 10
310  PRINT E(I);
320  NEXT I:PRINT
330  PRINT "FNX(X)=";FNX(10)
340  PRINT:PRINT
350 RETURN
```

```
run

A%= 213           B!= 74.7543    C#= 76346.432654302
A$="ABCDEFG"
E(X)=              0  1  2  3  4  5  6  7  8  9  10
FNX(X)= 115


A%= 0             B!= 0          C#= 0
A$=""
E(X)=              0  0  0  0  0  0  0  0  0  0  0
FNX(X)=
Undefined user function in 330
Ok
```

CLOSE

ステートメント

機能　ファイルを閉じる.

書式　CLOSE〔〔#〕〈ファイル番号〉〔,〔#〕〈ファイル番号〉〕・・・〕

文例　CLOSE

解説　〈ファイル番号〉に対応するファイルを閉じます. 以後, 指定された〈ファイル番号〉は, 異なるファイルを開くために利用できるようになります. また閉じられたファイルは, 同じあるいは異なったファイル番号によって再び開くことができます.

CLOSE 文では,〈ファイル番号〉を複数指定することにより, 一度に複数のファイルを閉じることができます. また〈ファイル番号〉が省略された場合には, そのとき, 開いているファイルすべてを閉じます.

CLOSE したファイルに対しては, 再び OPEN するまで入出力を行うことはできません.

CLOSE 文は, ファイルが出力用にオープンされていた場合, バッファに残っていたデータの掃き出しを行いますので, ファイルへの出力処理を正しく終了するためには, 必ず CLOSE を行わなくてはなりません.

END, NEW, SYSTEM, および R オプションなしの RUN の各文は, 自動的にすべてのファイルを閉じる処理を行います. これに対して, STOP 文はファイルを閉じる作業は行いませんので注意してください.

参照　OPEN, END, NEW, SYSTEM

サンプルプログラム

```
10 OPEN "ai" FOR OUTPUT AS #1
20 OPEN "scrn:" FOR OUTPUT AS #2
30 PRINT #1,"DISK"
35 CLOSE #1
40 PRINT #2,"SCREEN"
50 CLOSE #2


run
SCREEN
Ok
```

CLS
ステートメント

機能 画面をクリアする

書式 CLS〔〈数式〉〕

文例 CLS 0

解説 〈数式〉なしでCLS文を実行すると，CONSOLE文によって指定されたスクロールウィンドウの内部がクリアされます．

〈数式〉をつけて実行すると，画面全体がクリアされます．この場合，〈数式〉には，どんな数値を指定してもかまいません．

なお，画面をクリアした場合には，LPはビューポートの中央の点に移動します．

参照 SCREEN，COLOR，CONSOLE，VIEW

サンプルプログラム

```
100 CLS
110 SCREEN 1,0,0,0
120 LOCATE 1,1,0
130 DEFINT A-Z
140   FOR I=1 TO 10
150     FOR J=1 TO 50
160       IF RND<.5 THEN GOSUB MARU ELSE GOSUB MOJI
170     NEXT J
180    COLOR ,RND*15
190   CLS
200  NEXT I
210   COLOR 15,0:CLS
220 END
230 MARU
240     X=RND*720:Y=RND*512:R=RND*100:C1=RND*14+1:C2=RND*15
250     CIRCLE(X,Y),R,C1:PAINT(X,Y),C2.C1
260 RETURN
270 MOJI
280   COLOR RND*14+1.RND*15
290   X=RND*70+1 :Y=RND*24+1
300    LOCATE Y,X
310   PRINT "PC-100";
320 RETURN
```

COLOR

ステートメント

機　能　ディスプレイ画面のフォアグラウンドカラーおよびバックグラウンドカラーを指定する.

書　式　COLOR〔<フォアグラウンドカラー>〕〔, <バックグラウンドカラー>〕

文　例　COLOR 6, 0

解　説　画面のフォアグラウンドカラー，およびバックグラウンドカラーを設定します.<フォアグラウンドカラー> と <バックグラウンドカラー> は，いずれも〝パレット番号〟によって指定します.

<フォアグラウンドカラー>とは，画面にキャラクタや，点，線などを表示したりするときに使われる色のことです．N_{100}-BASICの種々のグラフィックコマンド(PSET，LINE，CIRCLEなど)で特別に色指定をしなかった場合，このフォアグラウンドカラーが採用されます.

<バッググラウンドカラー> とは，画面の色のことで，このステートメント実行後CLS文によって画面をクリアすると，この色によって画面が塗り変えられます.また以後PRESET文を色指定なしで実行すると，この色が採用されることになります.

初期状態では，COLOR15，0がセットされています.

参　照　文法編　カラー

サンプルプログラム

```
100 SCREEN 1,0,0,0:LOCATE 1,1,0
110 COLOR 15,0:CLS
120 GOSUB *LINEB
130 FOR I=1 TO 15
140  COLOR I:GOSUB *LINEB
150 NEXT I:END
160 *LINEB:PRINT "COLOR"
170 LINE(100,50)-STEP(400,300),,BF
180 RETURN
```

COMMON

ステートメント

機能　CHAIN 文が実行されたとき変数を引き渡す.

書式　COMMON <変数名>〔,<変数名>・・・〕

文例　COMMON A, B, XY(), NA$

解説　CHAIN 文が実行され，メモリ上のプログラムと外部からのプログラムがチェインされたとき，メモリ上のプログラムの変数を引き渡します．したがって COMMON 文は必ず CHAIN 文とペアになって使われます．

<変数名>は 1 行の許す範囲で何個でも並べることができますが，1 つのプログラム中の COMMON 文に同じ変数名があってはいけません．また配列変数名は〝()〟を付け加えて表現します．

配列変数をチェインした場合，チェインされる側のプログラム中で DIM 文により同じ配列変数名を改めて宣言すると，〝Duplicate Definition〟エラーとなります．また，配列の下限はそのまま引き渡されますので，チェインされる側で OPTION BASE 文によって配列の下限を変更しようとすると，やはり〝Duplicate Definition〟エラーとなります．

COMMON 文は非実行文ですからプログラム中のどこにあってもかまいませんが，少なくとも対応する CHAIN 文の前にあったほうが，プログラムがわかりやすくなってよいでしょう．

すべての変数を引き渡したいのであれば，CHAIN 文の ALL オプションを使う方が便利です．

参照　CHAIN, DIM, OPTION BASE

サンプルプログラム

```
100 DIM A(10)
110 COMMON B,C%,D#,E$,A()
120 A(5)=321
130 B=54
140 C%=654
150 D#=32.68666321#
160 E$="NEC PC-100"
170 CHAIN "TESTP"
180 END
run
A(5)= 321
B= 54
C%= 654
D#= 32.68666321
E=NEC PC-100
Ok
```

```
200 PRINT "A(5)=";A(5)
210 PRINT "B=";B
220 PRINT "C%=";C%
230 PRINT "D#=";D#
240 PRINT "E=";E$
```

COM(n) ON/OFF/STOP

ステートメント

機能　通信回線からの割込みを許可，保留する．

書式

COM(〈ポート番号〉) ON

COM(〈ポート番号〉) OFF

COM(〈ポート番号〉) STOP

文例　COM(1) ON

解説

コミュニケーションポート（RS-232C 回線）に外部からの通信が入ったことによる割込みを，許可（ON）するか，保留（OFF,STOP）するかを設定します．

〈ポート番号〉には，コミュニケーションポート番号を指定します．PC-100 の RS-232C 回線は 1 チャネルのみですから 1 を指定してください．

COM(n) ON 文は，〈ポート番号〉で指定されたポートの割込みを許可します．このステートメント実行後は，コミュニケーションポートに通信が入るごとに割込みを発生し，ON COM(n) GOSUB 文で定義されている処理ルーチンに分岐します．

COM(n) OFF 文，COM(n) STOP 文は，〈ポート番号〉で指定されたポートの割込みを保留します．このステートメント実行後は，通信があってもそのことを覚えているだけで処理ルーチンへの分岐は起こりません．しかし，その後 COM(n) ON 文によって割込みが許可されると，先程の通信によって処理ルーチンに分岐します．

なお，割込み処理ルーチンに制御が移ると自動的に COM(n) STOP（割込み保留）状態となりますので，割込み処理ルーチンの先頭で COM(n) OFF 文あるいは COM(n) STOP 文を実行する必要はありません．

注意：プログラム終了時には必ず COM(n) OFF または COM(n) STOP 文の状態にしておいてください．COM(n) ON 文は，RS-232C コミュニケーションポートを OPEN した後実行されなければ有効となりません．

参照　ON COM(n) GOSUB

サンプルプログラム

```
100 CLS
110 Y=13:X=1
120 OPEN "COM1:5E81NA" AS #1
130 ON COM(1) GOSUB RECIEVE
140 LOCATE 12,1:PRINT "受信"
150 LOCATE 1,1:PRINT "送信"
160 COM(1) STOP
170 A$=INKEY$:PRINT #1,A$;:PRINT A$;
180 COM(1) ON:GOTO 160
190 RECIEVE
200 X1=POS(0):Y1=CSRLIN
210 LOCATE Y,X
220 IF LOC(1)<>0 THEN PRINT INPUT$(LOC(1),#1);
230 X=POS(0):Y=CSRLIN
240 LOCATE Y1,X1
250 RETURN
```

CONSOLE

ステートメント

機能 画面のスクロールウィンドウの設定を行う.

書式 CONSOLE <スクロール開始行>, <スクロール行数>

文例 CONSOLE 1, 25

解説 <スクロール開始行> と <スクロール行数> を指定することによって，画面上でスクロールする領域（スクロールウィンドウ）を設定します.

どちらのパラメータも，1から，その画面の最大の行数までの範囲内の整数で，両方とも省略することはできません．ただし，ファンクションキーの表示がされているときは1行分，漢字入力テンプレートが設定されているときは2行分，表示可能な最大行数が減ることになりますのでご注意ください.

スクロールウィンドウが設定されると，以後画面に出力されるキャラクタはすべてこのウィンドウ内にのみ描かれ，ウィンドウの外にLOCATE文などでキャラクタを出力しようとすると“Illegal function call”エラーとなります.

ただし，グラフィックコマンドは，このスクロールウィンドウに影響されることはありません.

なお，引数なしのCLS文は，このスクロールウィンドウに対してのみ実行されます.

参照 WIDTH, WIDTH USING

サンプルプログラム

```
100 '-- scroll window --
110 CLS
120 CONSOLE 1,25
130 FOR I=1 TO 25 STEP 4
140 CONSOLE 1,I
150  GOSUB *PRINTSUB
160 NEXT I
170 END
180 *PRINTSUB
190  CLS
200  FOR J=1 TO 40
210   PRINT J
220   FOR K=0 TO 10:NEXT K
230  NEXT J
240 RETURN
```

```
5
6
7
8
9
10
11
12
```

CONT
コマンド

機能　ストップキー入力，またはSTOP文によって停止したプログラムの実行を再開する．

書式　CONT

文例　CONT

解説　CONTコマンドは通常デバッグのためにSTOP文と共に用います．STOP文や[STOP]キー([CTRL]+[C])の入力によりプログラムの実行を停止して，ダイレクトモードで変数の値などを調べたり変更した後に，CONTコマンドを入力すると，停止した箇所の次のステートメントから実行が再開されます．INPUT文からのプロンプトが出力されているときに停止した場合は，そのINPUT文から実行が再開されます．

ただし，実行停止中にプログラム内容の変更を行った場合には，CONTコマンドによる継続実行はできなくなります．

サンプルプログラム

```
100 FOR I=1 TO 15
110   PRINT "I=";I,"I^2=";I^2,"SQR(I)=";SQR(I)
120 NEXT

run
I= 1          I^2= 1        SQR(I)= 1
I= 2          I^2= 4        SQR(I)= 1.4142135623731
I= 3          I^2= 9        SQR(I)= 1.7320508075689
I= 4          I^2= 16       SQR(I)= 2
I= 5          I^2= 25       SQR(I)= 2.2360679774998
I= 6          I^2= 36       SQR(I)= 2.4494897427832

Break in 110
Ok
print I,I^3
 6             216
Ok
cont
I= 7          I^2= 49       SQR(I)= 2.6457513110646
I= 8          I^2= 64       SQR(I)= 2.8284271247462
I= 9          I^2= 81       SQR(I)= 3
I= 10         I^2= 100      SQR(I)= 3.1622776601684
I= 11         I^2= 121      SQR(I)= 3.3166247903554
I= 12         I^2= 144      SQR(I)= 3.4641016151378
I= 13         I^2= 169      SQR(I)= 3.605551275464
I= 14         I^2= 196      SQR(I)= 3.741657386774
I= 15         I^2= 225      SQR(I)= 3.8729833462074
Ok
```

COS

関数

機能　余弦値（コサイン）を返す.

書式　COS（<数式>）

文例　C=COS（3.1415926535898／2）

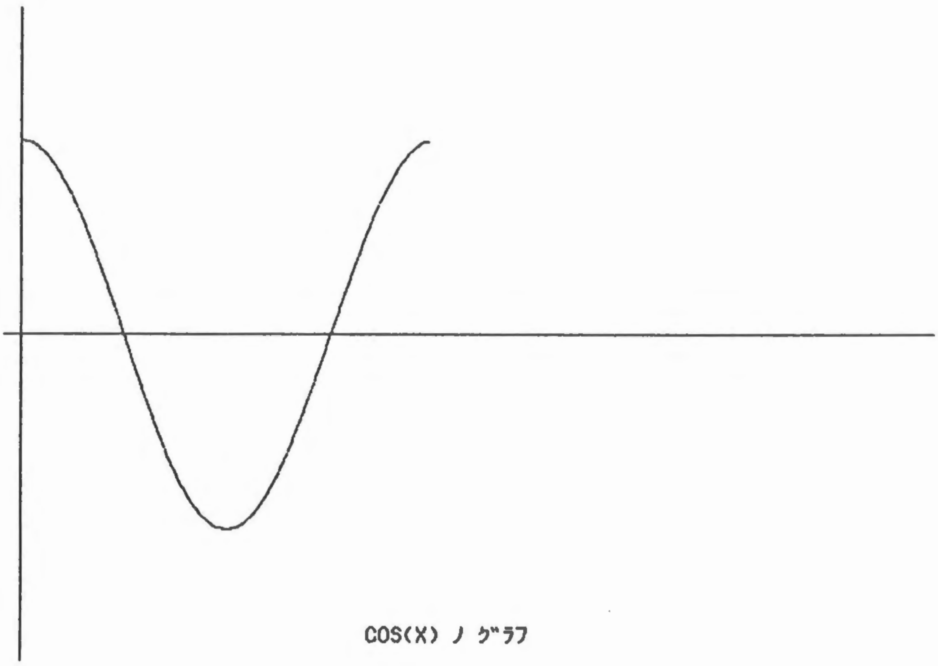
解説　<数式>の値に対する余弦値を返します. <数式>の単位はラジアンです.

また, <数式>にはどんな型の数値を指定しても, COS関数は倍精度の値を返します.

参照　SIN, TAN

サンプルプログラム

```
100  CLS:SCREEN 1 :KEY OFF
110  WINDOW (-5,100)-(300,-100)
120   LINE (-5,0)-(300,0),6
130   LINE (0,100)-(0,-100),6
140 'COS ｸﾞﾗﾌ
150   LOCATE 25,30,0:PRINT "COS(X) ﾉ ｸﾞﾗﾌ ";
160  FOR X= 0 TO 12.5663 STEP .01
170    Y=COS(X):GOSUB *POINTSET
180  NEXT
190    BEEP
270  LOCATE 1,1
280 END
290 *POINTSET
300   PSET (X*20,Y*60),10
310  RETURN
```



CSNG
関数

機能	整数値，倍精度実数値を単精度実数値に変換した値を返す．
書式	CSNG（<数式>）
文例	A!=CSNG（B#）
解説	<数式>の値を有効数字6桁の単精度実数型に変換します．その値が−9.99999E+62〜9.99999E+62の範囲にない場合は，〝Overflow〞エラーが起こります．
参照	CINT，CDBL

サンプルプログラム

```
100 REM ｼｮｳｽｳﾃﾝｲｶ ﾆﾝｲ ﾉ ｹﾀ ﾃﾞﾉ 4ｼｬ5ﾆｭｳ
110 A#=1.23456789#:D=1:GOSUB 140:GOSUB 170
120 A#=1.23456789#:D=5:GOSUB 140:GOSUB 170
130 END
140 IF A#=INT(A#) THEN AD#=A#:RETURN
150 AD#=INT(A#*10^D+.5)/10^D
160 RETURN
170 PRINT A#,CSNG(AD#)
180 RETURN
```

```
run
 1.23456789     1.2
 1.23456789     1.23457
Ok
```

CSNG$

関数

機　能　2バイトキャラクタを1バイトキャラクタに変換する.

書　式　CSNG$（<文字列>）

文　例　S$=CSNG$（D$）

解　説　<文字列> で与えられる英数字の2バイトキャラクタを同じ表現の1バイトキャラクタに変換します.

<文字列> には英大文字，英小文字，数字の2バイトキャラクタを指定します.

<文字列>に上記以外の2バイトキャラクタ，または1バイトキャラクタを指定した場合，CSNG$関数はこれらの文字を1バイトキャラクタには変換しません.

1バイトキャラクタ，2バイトキャラクタについては，付録ASCIIコード表，漢字コード表をご覧ください.

1バイトキャラクタを2バイトキャラクタに変換したいときにはCDBL$関数を使います.

参　照　CDBL$

サンプルプログラム

```
100 SCREEN 0,0,0,1
110 A$="日本電気　ＰＥＲＳＯＮＡＬ　ＣＯＭＰＵＴＥＲ ＰＣ－
１００"
120 PRINT A$
130 PRINT CSNG$(A$)

run
日本電気　ＰＥＲＳＯＮＡＬ　ＣＯＭＰＵＴＥＲ ＰＣ－１００
日本電気　PERSONAL　COMPUTER PC-100
Ok
```

CSRLIN
関数

機 能 現在のカーソルの行位置を返す.

書 式 CSRLIN

文 例 Y=CSRLIN

解 説 現在のカーソルの行位置（垂直方向）を返します.

返される値の範囲は，1から，WIDTH 文によって指定された最大の行数までで，値が小さいほど画面の上の方を意味します.

なお，カーソルの桁位置（水面方向）を知るためには，POS 関数を使用します.

参 照 POS

サンプルプログラム

```
100 SCREEN 1,0,0,0:CLS
110 FOR I=1 TO 21
120 IF CSRLIN=10 THEN COLOR 13:PRINT "PC-100"
130 COLOR 15:PRINT "NEC"
140 NEXT
```

CVI/CVS/CVD
関数

機 能	文字列を数値データに変換する.

書 式

CVI (<2バイトの文字列>)
CVS (<4バイトの文字列>)
CVD (<8バイトの文字列>)

文 例

A%=CVI (A$)
B!=CVS ("A3Bd")
C#=CVD (NUM$)

解 説

ディスク上のランダムファイルから読み込んだデータは文字型になっているため, CVI, CVS, CVD 各関数を使って, 数値データに変換しなければなりません.

CVIは2バイトの文字列 (即ち16ビットのデータ) を整数値に変換します.

CVSは4バイトの文字列 (即ち32ビットのデータ) を単精度実数値に変換します.

CVDは8バイトの文字列 (即ち64ビットのデータ) を倍精度実数値に変換します.

実際の文字列と変換後の数値との関係は, わかりやすく整数型で説明すると次のようになります.

A$= "AB" の場合, 即ち, A$=CHR$ (&H41)+CHR$ (&H42) の場合には, CVI (A$) は, &H4241 (=16961) になります (バイトが前後逆になることに注意). つまり, キャラクタコードを数値データとみなすわけです.

CVS, CVDの場合も, これと同様にして, 各キャラクタコードを正規化された浮動小数点形式の数値データとみなすことになります.

なお, これとは逆に, 数値データをランダムファイルに格納するときは, MKI$, MKS$, MKD$を使って文字型データに変換しなくてはなりません.

参 照

MKI$／MKS$／MKD$

サンプルプログラム

```
100 OPEN "RDATA" AS #1
110 FIELD #1,8 AS A$
120 GET #1,1:A=CVI(A$):PRINT A
130 GET #1,2:A=CVS(A$):PRINT A
140 GET #1,3:A=CVD(A$):PRINT A
150 END
160 'このプログラムで読むデータはMKI$／MKS$／ MKD
$のサンプルプログラムで作られます。
```

```
run
 987
 132.456
 123.456789123
Ok
```

DATA
ステートメント

機能 READ 文で読み込まれるデータを格納する.

書式 DATA 〈定数〉〔, 〈定数〉・・・〕

文例 DATA 1, CBA, 1465

解説 DATA 文は非実行文で, プログラム中のどこに置いてもかまいません. 1つの DATA 文には, 1行（255文字）に入るだけのデータをセットすることができます. また, 1つのプログラム内には, 任意の数の DATA 文を置くことができます. DATA 文中のデータは, 1つのプログラム内で, 1つの連続した項目並びとみなされます.

READ 文は行番号の小さい方から順番に, DATA 文中のデータを読み込んでいきます.
〈定数〉は, 任意の型の定数, 即ち, 整数, 単精度実数, 倍精度実数, 文字定数のいずれでもかまいません. ただし, 定数式（2 * 3 など）は許されません. また READ 文で読み込む場合, READ 文で指定されている変数の型は, 対応する DATA 文の定数の型と一致しなければなりません. さもないと "Syntax error"（構文エラー）となります.

ただし, DATA 文中のデータが数値である場合は, READ 文で読み込む変数の型は何でもよく, 読み込まれた時点でそのデータはその変数の型となります.

DATA 文中のデータはカンマ（,）によって区切られますが, データが文字定数の場合, その文字列中にカンマ（,）, ピリオド（.）, および意味のある空白を入れたいときは, その文字定数の前後をダブルクォーテーション（"）でくくる必要があります.

RESTORE 文によって READ 文で読み込む DATA 文を行単位で指定することができます.

参照 READ, RESTORE

サンプルプログラム

```
100 DATA 10,10,    NEC PC-100,"    NEC PC-100"
110 READ A
120 READ B$:B=VAL("&H"+B$)
130 READ C$:READ D$
140 PRINT A:PRINT B
150 PRINT C$:PRINT D$
```

```
run
 10
 16
NEC PC-100
   NEC PC-100
Ok
```

DATE$

システム変数

機　能　日付を設定する，あるいは日付を返す．

書　式　DATE$

文　例　D$=DATE$

解　説　DATE$には常に現在の日付が mm／dd／yyyy（〝01／01／1984〃等）の形で入れられており，いつでもその内容を見ることができます．他の変数に代入した場合も，同じ形の文字列が代入されます．

またDATE$には，日付をセットすることもできます．セットには，次のどの形式を用いてもかまいません．

DATE$＝〝mm—dd—yy〃
DATE$＝〝mm／dd／yy〃
DATE$＝〝mm—dd—yyyy〃
DATE$＝〝mm／dd／yyyy〃

月，日に1桁のみを指定すると，その前にゼロ(0)が付加されます．年に1桁のみを指定すると200X年が仮定されます．また年に2桁のみを指定した場合，80～99の指定に対しては19XX年が仮定されます．年に4桁を指定する場合は1980～2099年まで指定することができます．

注意：PC-100は，DOSにより常に日付が更新され，正しい日付が維持されるようになっています．従って不用意に日付を変えると，せっかくの正しい日付がこわれることになりますのでご注意ください．

参　照　TIME$，TIMER

サンプルプログラム

```
100 ' "12-25-83" --> "25,Dec.1983"
110 DA$=DATE$
120  DIM MONTH$(12)
130  FOR I=1 TO 12
140   READ MONTH$(I)
150  NEXT I
160  DATE$="12-25-83"
170  GOSUB *TRNS
180  PRINT DATE$;" --> ";M$
190 DATE$=DA$
200 END
210 *TRNS
220  M$=MID$(DATE$,4,2)+","
230  M$=M$+MONTH$(VAL(LEFT$(DATE$,2)))
240  M$=M$+".19"+RIGHT$(DATE$,2)
250 RETURN
260  DATA Jan,Feb,Mar,Apr,May,Jun
270  DATA Jul,Aug,Sep,Oct,Nov,Dec
```

```
run
12-25-1983 --> 25,Dec.1983
Ok
```

DEF FN
ステートメント

機 能	ユーザー関数を定義する.

書 式　DEF FN<名前>〔(<パラメータリスト>)〕= <関数の定義式>

文 例　DEF FNA(X, Y)=X * 2 +Y * 3

解 説　<名前> は変数名として正しい形をしたものでなくてはなりません. 関数の名前は "FN" とこれに続く <名前> を含めたものとなります. 関数が返す値の型は, この <名前> の型によって決まります.

<パラメータリスト> はその関数が呼ばれたときに, <関数の定義式> 中の同じ名前の変数に対応します. この変数名は, <関数の定義式>を定義する際にのみ有効となりますので, プログラム中に同一名の変数があっても影響はありません. また, <関数の定義式> 中で使われている変数が <パラメータリスト> 中にない場合は, DEF FN 文はその名前を持つ変数がその時点で持っている値を使います.

<関数の定義式>は, その関数の演算内容を記述する式で, 1行の範囲に限られます. この定義式は, 数値関数, 文字関数, その混在のいずれでもかまいませんが, <パラメータリスト>内の変数と, それに渡される変数とが同一の型でなければなりません.

DEF FN 文は非実行文ですが, これによって定義される関数がプログラム中で呼ばれる前に実行されていなければなりません. もし, 定義される前にユーザー関数が呼ばれると, "Undefined user function" エラーが生じます.

注意 : DEF FN 文は, ダイレクトモードで実行することはできません.

参 照　文法編　変数名と型宣言文字

サンプルプログラム

```
100 DEF FNFUKURI(G,R,N)=G*(1+R)^N
110 DEF FNTANRI(G,R,N)=G*(1+R*N)
120 INPUT "元金    :",GANKIN
130 INPUT "年利    :",NENRI:INPUT "年数    :",NEN
140 FGOKEI=FNFUKURI(GANKIN.NENRI.NEN)
150 TGOKEI=FNTANRI(GANKIN,NENRI,NEN)
160 F$="¥¥##########,._-"
170 PRINT "複利では ";USING F$;FGOKEI
180 PRINT "単利では ";USING F$;TGOKEI
run
元金    :150000
年利    :.065
年数    :3
複利では       ¥181,192.-
単利では       ¥179,250.-
Ok
```

DEFINT/SNG/DBL/STR

ステートメント

機 能　変数の型宣言をする.

書 式

```
DEF | INT | 〈文字の範囲〉〔,〈文字の範囲〉・・・〕
    | SNG |
    | DBL |
    | STR |
```

文 例　DEFINT A, I−K

解 説　〈文字の範囲〉で指定された文字で始まる変数名の変数を，DEFINT は整数型に，DEFSNG は単精度型に，DEFDBL は倍精度型に，DEFSTR は文字型にそれぞれ定義します.

〈文字の範囲〉で指定される文字は英字（A−Z）1文字で，複数個指定する場合はマイナス記号（−）でつないでその範囲を示します.

ただし，これらによって行われた型宣言より，型宣言文字（%，!，#，$）による指定の方が優先されます.

なお，型宣言が行われていない文字で始まる変数は，すべて倍精度変数とみなされます（変数Aと変数A#は同一の変数となります）.

サンプルプログラム

```
100 DEFINT I-M
110 DEFSNG A,B
120 DEFDBL D
130 I1=1.23:J=65.643
140 ABC=1.23:BBB=65.643
150 D=3.141592654#
160 D%=3.1415926540003#
170 PRINT "I1=";I1,,"J=";J
180 PRINT "ABC=";ABC,,"BBB=";BBB
190 PRINT "D=";D,"D%=";D%
200 END
```

```
run
I1= 1                          J= 66
ABC= 1.23                      BBB= 65.643
D= 3.141592654                 D%= 3
Ok
```

DEF SEG

ステートメント

機能　セグメントのベースアドレスを設定する.

書式　DEF SEG〔=〈セグメントアドレス〉〕

文例　DEF SEG=&H0100

解説　〈セグメントアドレス〉は，&H0 から&HFFFF までの値で指定しますが，実際のアドレスはこの値を16倍，即ち 4 ビット左にシフトした&H0 から&HFFFF0番地を指します．初期値は，BASIC のデータセグメント（DS）となっています．

=〈セグメントアドレス〉を省略すると，セグメントは BASIC のデータセグメントに設定されます．

セグメントを変更する DEF SEG 文が実行されると，RUN コマンドが実行されても，その値は変化しません．

N_{100}-BASIC は，最高 1MB(=1024キロバイト=1048576バイト)まで利用できますが，機械語命令は65535（&HFFFF）番地までしか指定できませんので，プログラム領域を複数のセグメントに分けてメモリを活用するようにしています．そこで，PEEK 文や POKE 文によりメモリの内容を直接のぞいたり書き込んだりする場合には，この DEF SEG 文によってセグメントベースを指定することが必要となります．いったんセグメントベースが指定されれば，PEEK 文あるいは POKE 文ではオフセット値，つまりベースアドレスからの増分値を指定することにより実際のメモリ上のアドレスにアクセスすることができるわけです．

メモリ配置については BASIC ユーザーズガイドを参照してください．

注意：DEF SEG 文は，DEF と SEG の間に必ずスペースを入れなければなりません．スペースを省略して DEFSEG とすると変数として扱われます．

また，BASIC は，指定されたセグメントアドレスが有効かどうかのチェックを行いませんのでご注意ください．

参照　BLOAD, BSAVE, CLEAR, POKE, PEEK, VARPTR, VARPTR$

DEF USR
ステートメント

機 能　**機械語で作られたユーザー関数の実行開始アドレスを定義する.**

書 式　DEF USR〔〈番号〉〕=〈開始アドレス〉

文 例　DEF USR 3 =&HF000

解 説　USR 関数（ユーザー関数）が呼び出す，機械語ルーチンの実行開始アドレスの設定を行います.

〈番号〉は，0〜9までの値で，複数のユーザー関数を用いる場合の識別を行います．したがって最大10個までのユーザー関数を準備，利用することができます．〈番号〉が省略された場合には 0 と解釈されます.

〈開始アドレス〉は〈番号〉によって指定されるユーザー関数の実行開始アドレスを，直前に実行されたDEF SEG 文で指定されるセグメントベースからのオフセット値で指定します．一般にセグメントのベースは，CLEAR 文の〈メモリの上限〉で指定したものとします.

USR 関数の作り方については，BASIC ユーザーズガイドを参照してください.

参 照　USR

DELETE

コマンド

機　能　プログラムの部分削除を行う．

書　式　DELETE〔<始点行番号>〕〔－〔<終点行番号>〕〕

文　例　DELETE 10－250

解　説　<始点行番号> から <終点行番号> までのプログラムを削除しますが，この 2 つのパラメータはそれぞれ省略することもできます．以下にいろいろなパラメータの指定方法を挙げます．

DELETE 10

行番号10を削除します．

DELETE 50－100

行番号50から行番号100までを削除します．

DELETE 50－

行番号50からプログラムの終わりまでを削除します．

DELETE －100

プログラムの先頭から行番号100までを削除します．

なお DELETE 文では，行番号を指定する際，現在行を表すピリオド（.）を使用することができます．

サンプルプログラム

```
list
100 ' DELETE ｺﾏﾝﾄﾞ ﾉ ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ
110 '
120 ' DELETE ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｦ
130 ' ｻｸｼﾞｮ ｼﾏｽ.
140 '
150 PRINT " DELETE ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｦ"
160 PRINT " ｻｸｼﾞｮ ｼﾏｽ."
170 END
Ok
delete 110-130
Ok
```

```
list
100 ' DELETE ｺﾏﾝﾄﾞ ﾉ ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ
140 '
150 PRINT " DELETE ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｦ"
160 PRINT " ｻｸｼﾞｮ ｼﾏｽ."
170 END
Ok
run
 DELETE ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｦ
 ｻｸｼﾞｮ ｼﾏｽ.
Ok
```

DIM

ステートメント

機能　配列変数の大きさを指定し，メモリ領域に割り当てる．

書式　DIM 〈変数名〉(〈添字の上限〉〔, 〈添字の上限〉...〕), ・・・

文例　DIM A (12, 2), B (3, 4)

解説　〈変数名〉には，定義したい配列変数の名前を指定します．〈添字の上限〉には，配列変数の添字がとる上限(即ち次元)を設定します．DIM 文はメモリ上にその配列の領域を割り当てます．DIM 文で宣言せずに配列変数を用いた場合，その添字の上限は10とみなされます．もし，設定された上限より大きい値の添字が用いられた場合は，"Subscript out of range" エラーが起こります．

添字の下限は，OPTION BASE 文によって，0か1に指定することができますが，何も指定しなければ，つねにゼロ（0）です．
DIM 文では，配列変数をカンマ (,) で区切って並べることにより，複数定義することができます．

DIM 文が実行されると，その配列のすべての要素の値は 0（文字型の場合はヌルストリング）に設定されます．

配列の次元数の上限は，255です．また，数値配列の場合，1個の配列に対し指定できる要素の数は次のとおりです．ただし，これらの数はメモリの大きさによっても制限されますので，ご注意ください．

整数型の場合：32767個
単精度実数型の場合：16383個
倍精度実数型の場合：8191個

なお，同名の配列変数を再度宣言しようとすると，"Duplicate Definition" エラーが生じます．しかし，ERASE 文か CLEAR 文を用いて配列領域をクリアすれば，再度同名の配列変数を使用することができます．

参照　OPTION BASE

サンプルプログラム

```
100 ' ｸｸ ﾉ ﾋｮｳ ｦ ﾂｸﾙ
110 OPTION BASE 1
120 DIM KUKU(9,9)
130 FOR I=1 TO 9:FOR J=1 TO 9
140   KUKU(I,J)=I*J
150 NEXT J,I
155 PRINT "掛算 九九 の 表 "
160 FOR I=1 TO 9:FOR J=1 TO 9
170   PRINT USING "####";KUKU(J,I);
180  NEXT J:PRINT
190 NEXT I
200 END
```

```
run
掛算 九九 の 表
   1   2   3   4   5   6   7   8   9
   2   4   6   8  10  12  14  16  18
   3   6   9  12  15  18  21  24  27
   4   8  12  16  20  24  28  32  36
   5  10  15  20  25  30  35  40  45
   6  12  18  24  30  36  42  48  54
   7  14  21  28  35  42  49  56  63
   8  16  24  32  40  48  56  64  72
   9  18  27  36  45  54  63  72  81
Ok
```

DRAW
ステートメント

機能	ディスプレイ画面上に図形を描く．
書式	DRAW 〈文字列〉
文例	DRAW "L10U5R10D5"

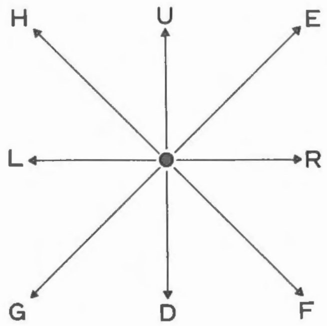
解説

グラフィックマクロ命令をサポートするステートメントです．グラフィックマクロ命令は文字列で表される描画用コマンドで，基本的には，参照点を相対的に移動しながら線画を描いて行くものです．

グラフィックマクロ命令は大きく分けて，移動，回転，カラー，スケールファクタ，ローカルマクロ，の５種類のコマンド体系から成っています．

移動コマンドには，次のようなものがあります．

U 〈距離〉 ……………上方向へ移動します．
R 〈距離〉 ……………右方向へ移動します．
D 〈距離〉 ……………下方向へ移動します．
L 〈距離〉 ……………左方向へ移動します．
E 〈距離〉 ……………右上方向へ移動します．
F 〈距離〉 ……………右下方向へ移動します．
G 〈距離〉 ……………左下方向へ移動します．
H 〈距離〉 ……………左上方向へ移動します．



Mx，y…………………X 方向に x，Y 方向に y だけ移動(絶対指定)します．x，y の前にプラス(＋)またはマイナス(－)がつくと相対指定となります．

上の移動コマンドの前に付加して使用するサブコマンドがあります．

B………………………移動のみで描画しません．
N………………………描画後も LP（最終参照点）を始点のままとします．
N サブコマンドを指定しないときは，LP は常に終点に移動します．

回転コマンドには，次のようなものがあります．

A 〈式〉 ………………移動コマンドで描画するべき図形（相対座標指定のものに限る）を，回転させて表示します．
〈式〉の値が 0 ならば 0 度，1 ならば90度，2 ならば180度，3 ならば270度，回転します．
一度指定すると，再度指定するまで有効です．

TA 〈角度〉 …………移動コマンドで描画するべき図形（相対座標指定のものに限る）を，〈角度〉

だけ回転させて表示します．
〈角度〉には，−360〜360の値を指定します．正の値であれば反時計廻り，負の値であれば時計廻りに回転します．

カラーコマンドには，次のようなものがあります．

C 〈パレット番号〉 … 〈パレット番号〉で指定された色で描画します．省略時にはフォアグラウンドカラーに設定されます．

P 〈領域色〉，〔〈境界色〉〕
指定された境界色で囲まれた領域を，指定された色で塗ります．
このコマンドでタイルペイントすることはできません．

スケールファクタコマンドは，次のとおりです．

S 〈式〉 ………………スケールファクタを設定します．〈式〉は1〜255の整数で，その 4 分の 1 の値がスケールファクタとなります．移動コマンドで指定される移動距離に，このスケールファクタを乗じたものが実際の移動距離となります．省略時には 4 が設定されます．
一度指定すると，再度指定するまで有効です．

ローカルマクロコマンドは，次のとおりです．

X 〈文字変数〉； ……〈文字変数〉中に代入されているグラフィックマクロ命令を実行します．〈文字変数〉には，グラフィックマクロ命令の文字列をあらかじめ入れておきます．〈文字変数〉の後のセミコロン(；)は，必ず入れてください．
ローカルマクロコマンドは，繰り返し図形を描くときなどに便利です．

グラフィックマクロ命令では，〈距離〉，〈角度〉，〈式〉などのパラメータに，ふつう定数を指定しますが，〝＝〈変数名〉；〟の形にすれば，数値変数を指定することもできます．この場合，〈変数名〉の後ろにはセミコロン(；)を必ず入れなければなりません．こうすることにより，〈変数名〉の値をプログラム上で操作でき，めんどうな文字列演算を行うことなくグラフィックマクロ命令をコントロールすることができます．

なお，各コマンドの区切りとしてセミコロン（；）を使用してもさしつかえありません．

DRAW 文と，ほかのグラフィックコマンドを混ぜて使ってもかまいませんが，スケールファクタの値などによく注意しないと予期しない図形を描いてしまいます．

また，DRAW 文でサポートできる座標系は，スクリーン座標系のみですのでご注意ください．

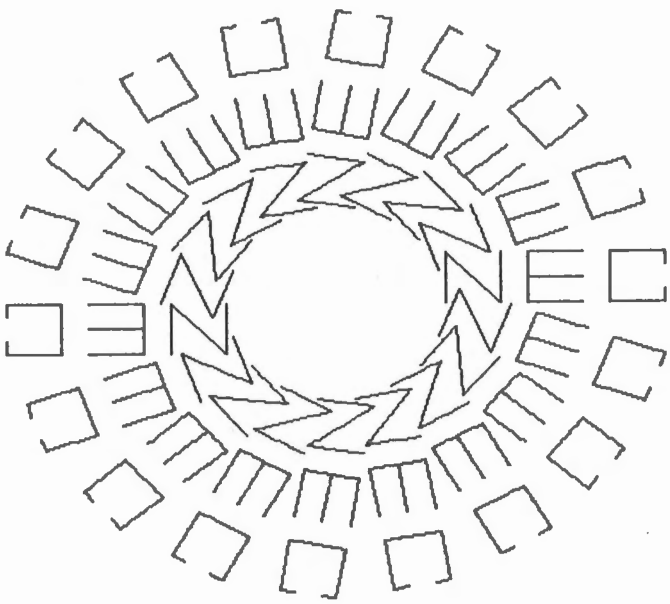
なお，DRAW 文によるカラー (C) とスケールファクタ (S) の指定は，CLEAR 文により初期状態

にリセットされます．

サンプルプログラム

```
100 CLS:LOCATE 1,1,0
110 FOR I=0 TO 360 STEP 20
120 J=J+1
125 CO=J MOD 8+7
130 DRAW"bm360,256"
140 DRAW"c=co;"
150 DRAW"ta=i;"
160 DRAW"s16br16U8F8U8"
170 DRAW"br4nr8d4nr8d4r8"
180 DRAW"br4nr8u8r8d2bd4d2"
190 NEXT
```

Ok



DSKF
関数

機能　ディスクの諸元を返す.

書式　DSKF (〈ドライブ番号〉〔,〈機能〉〕)

文例　PRINT DSKF(1)

解説　〈ドライブ番号〉で指定されたディスクに関する情報を関数の値として返します.〈ドライブ番号〉には, A, B, C, D～に対応して1, 2, 3, 4～を指定しますが, 0を指定しますとカレントドライブを指定したものとみなします.

〈機能〉は値として返す情報の種類を指定します.〈機能〉として指定する値と, その時に返される関数の値の意味は次のようになっています.

省略時：ディスクの残り容量をクラスタ単位で返す.
0：ドライブ当りのクラスタ数を返す.
1：セクタ当りのバイト数を返す.
2：クラスタ当りのセクタ数を返す.

サンプルプログラム

```
100 INPUT "Drive name";DR$
110 DR=INSTR("ABCD",DR$)
120 IF DR=0 THEN 100
130 PRINT "Track  =";DSKF(DR,0);"/drive"
140 PRINT "Sector =";DSKF(DR,1);"/track"
150 PRINT "Disk space =";DSKF(DR)
```

```
run
Drive name? A
Track  = 354 /drive
Sector = 512 /track
Disk space = 186
Ok
```

DSKI$

関数

機　能　ディスクからデータを直接読み出す．

書　式　DSKI$(〈ドライブ番号〉，〈トラック番号〉，〈セクタ番号〉，〈メディアタイプ〉〔，〈セクタ長〉，〈セクタ数／トラック〉〕)

文　例　D$＝DSKI$（2，1，1，1，2，9）

解　説　通常のファイル操作（OPEN，CLOSEなど）とは無関係に，ディスク上の指定したセクタから直接読み出しを行います．

DSKI$は，そのパラメータにより指定されたセクタ上に置かれているデータを0番バッファに読み出すとともに，最初の255文字までの文字列を関数の値として返します．

文字列の長さは255文字までしか許されないために，関数の値としてデータすべてを得ることはできません．そこで，通常のランダムアクセスの場合と同じようにFIELD文によって0番バッファへ複数の文字列変数を割り当てることにより，これを解決することができます．あるいは，VARPTR関数により，0番バッファの置かれているメモリ番地を調べた上でPEEK関数によりバッファからデータを読み出すこともできます．

なお，0番バッファの大きさは，BASIC起動時に／S：パラメータにより，必要な分だけ指定しておかなければなりません（省略時は128バイトです）．例えばセクタ長が1024バイトのディスクをDSKI$で読むには，DOSからBASICを呼び出すときに，

```
BASIC／S：1024
```

としなければなりません．

〈ドライブ番号〉は，次のように指定します．

1：Aドライブ
2：Bドライブ

〈トラック番号〉は，次のように指定します．

0～79：両面ディスク
0～39：片面ディスク

〈セクタ番号〉は，次のように指定します．

1～8：8セクタ／トラックのディスク
1～9：9セクタ／トラックのディスク

〈メディアタイプ〉は，次のように指定します．

0：片面倍密度
1：両面倍密度

〈セクタ長〉は，次のように指定します．省略時には，2が仮定されます．

1：256バイト

2：512バイト

3：1024バイト

4：2048バイト

5：4096バイト

〈セクタ数／トラック〉は，次のように指定します．省略時には，9が仮定されます．

8：8セクタ／トラックのディスク

9：9セクタ／トラックのディスク

複数のディスクが接続されている場合，それぞれのディスクの諸元についてはDSKF関数により調べることができます．

注意：この関数は，フロッピーディスクに対してのみ有効です．

参照　DSKO$, FIELD, DSKF, VARPTR

サンプルプログラム

```
100 'ＢＡＳＩＣ起動時に、“ＢＡＳＩＣ／Ｓ：５１２”とし
てください
110 FIELD #0,128 AS A$,128 AS B$,128 AS C$,128 AS D$
120 FOR TR=0 TO 79
130 FOR SE=1 TO 8
140 DU$=DSKI$(1,TR,SE,1)
150 A1$=A$:B1$=B$:C1$=C$:D1$=D$
160 FOR I=1 TO 128
170 J$=MID$(A1$,I,1):IF J$<>"" THEN IF ASC(J$)<32 THEN
MID$(A1$,I,1)=" "
180 K$=MID$(B1$,I,1):IF K$<>"" THEN IF ASC(K$)<32 THEN
MID$(B1$,I,1)=" "
190 L$=MID$(C1$,I,1):IF L$<>"" THEN IF ASC(L$)<32 THEN
MID$(C1$,I,1)=" "
200 M$=MID$(D1$,I,1):IF M$<>"" THEN IF ASC(M$)<32 THEN
MID$(D1$,I,1)=" "
210 NEXT
220 PRINT "ﾄﾗｯｸ=";TR,"ｾｸﾀ=";SE:PRINT:PRINT A1$;B1$;C1$;
D1$:PRINT
230 NEXT SE:NEXT TR
```

```
ﾄﾗｯｸ= 0        ｾｸﾀ= 6

SYSTEM#FILE          C;&      IO      SYS             ×  MSDOS   SYS
     朽    ﾄD  COMMAND COM             g▮  2 VB
TELCOM  EXE          循& C  ?  GW      EXE       慶( S      FILCON  EXE
     i'aY  脱  SETUP   COM              ｻ く"
BASIC   EXE          ﾇj) ﾄ     INP     BAS       彬*   "    CONFIG  SYS
     中"   B   HRDDRV  SYS             盾    ｾ
MUMUMU  BAS          僖, " ﾋ   PENGIN  BAS       + *   ｧ    ｵﾊﾞQ     BAS
     9 *   D   TAILINGUBAS             Z *

ﾄﾗｯｸ= 0        ｾｸﾀ= 7

おえかきBAS          o *   "   PALETTE BAS         *   I    PUSING  BAS
     *   .   CDBL    BAS             Nm* ! ﾄ
肘                    )* "     DEBUG   COM       杯a # z/   CHKDSK  COM
     oa /    SORT    BAS             撃* 6
瀕S     BAS          P   7 :   BAS     BAS       泥* 8 9    百SS     BAS
     ｺ   9      募募募募募募募募募募募募募募募
 募募募募募募募募募募募募募募募  募募募募募募募募募募募募募募募  募募募募募募募
募募募募募募募募  募募募募募募募募募募募募募募募
```

DSKO$

ステートメント

機　能　ディスクにデータを直接書き込む．

書　式　DSKO$　〈ドライブ番号〉，〈トラック番号〉，〈セクタ番号〉，〈メディアタイプ〉〔，〈セクタ長〉，〈セクタ数／トラック〉〕

文　例　DSKO$ 2，11，3，1，2，9

解　説　通常のファイル操作とは無関係に，ディスク上の指定したセクタに直接書き込みを行います．この文は，既存のディスクファイルを壊す恐れがありますので，ディスク及びファイルの構成を正しく理解した上で用いてください．これらについては，MS-DOS プログラマーズリファレンスマニュアルで解説されています．

DSKO$は，そのパラメータにより指定されたセクタに，0番バッファの内容を書き込みます．0番バッファは，通常のディスクファイル操作の際には用いることはできませんが，DSKI$とDSKO$による直接操作の際にのみ，ユーザーが用いることができます．

書き込むデータの準備は，ランダムアクセスの場合と同じように FILED 文により，0番バッファへ変数領域を割り付けた後，LSET，RSET により行います．あるいは，VARPTR 関数により，0番バッファの置かれているメモリ番地を調べた上で，POKE 文によりバッファ中にデータを準備することもできます．

なお，0番バッファの大きさは，BASIC 起動時に／S：パラメータにより，必要な分だけ指定しておかなければなりません（省略時は128バイトです）．

〈ドライブ番号〉は，次のように指定します．

1：Aドライブ
2：Bドライブ

〈トラック番号〉は，次のように指定します．

0～79：両面ディスク
0～39：片面ディスク

〈セクタ番号〉は，次のように指定します．

1～8：8セクタ／トラックのディスク
1～9：9セクタ／トラックのディスク

〈メディアタイプ〉は，次のように指定します．

0：片面倍密度
1：両面倍密度

〈セクタ長〉は，次のように指定します．省略時には，2が仮定されます．

1：256バイト

2：512バイト

3：1024バイト

4：2048バイト

5：4096バイト

〈セクタ数／トラック〉は，次のように指定します．省略時には，9が仮定されます．

8：8セクタ／トラックのディスク

9：9セクタ／トラックのディスク

複数のディスクが接続されている場合，それぞれのディスクの諸元については DSKF 関数により調べることができます．

注意：このステートメントは，フロッピーディスクに対してのみ有効です．

参照　DSKI$, FIELD, DSKF, VARPTR

サンプルプログラム

```
100 '------    Write to the disk    -------
110 'このプログラムはｄｉｓｋに書き込みますから
120 '                    不用意に使用しないでください
130 'このプログラムはＢＡＳＩＣ起動時に
140 '              “ＢＡＳＩＣ／Ｓ：５１２”としてください
150 FIELD #0,128 AS A$,128 AS B$,128 AS C$,128 AS D$
160 LSET A$=STRING$(128,CHR$(FF))
170 LSET B$=STRING$(128,CHR$(FF))
180 *START
190   INPUT "Ｄｒｉｖｅ　ｎａｍｅ？",DR$
200   DR=INSTR("abcdef",DR$)
210   IF DR<1 OR DR>5 THEN *START
220   INPUT "片面／両面（０／１）？？",ME
230   IF ME<0 OR ME>1 THEN 220
240 *REENT
250   INPUT "Ｔｒａｃｋ，ｓｅｃｔｏｒ",TR,SE
260   IF  TR<0 OR TR>79 THEN *REENT
270   IF SE<1 OR SE>9 THEN *REENT
280  DSKO$ DR,TR.SE.ME
290 END
```

```
run
Ｄｒｉｖｅ　ｎａｍｅ？a
片面／両面（０／１）？？1
Ｔｒａｃｋ，ｓｅｃｔｏｒ2,5
Ok
```

EDIT

コマンド

機　能　指定された行を画面上へ表示し，カーソルを移動する．

書　式　EDIT |〈行番号〉|
　　　　　　　　　　|　.　　　|

文　例　EDIT 30

解　説　画面上での編集ができるように，指定された行を表示し，カーソルをその行の先頭に置きます．エラー発生時や，STOP 文を実行した直後に〝EDIT　.″を行うと，エラーの発生した行，あるいは実行した STOP 文を含む行が表示され，ただちに修正作業に入ることができます．

スクリーンエディタの詳細については，BASIC ユーザーズガイドをご覧ください．

P オプション付きでセーブされたファイルをロードして EDIT コマンドにより，内容を表示，変更しようとした場合には，〝Illegal function call″エラーとなります．

参　照　LIST，DELETE，SAVE

サンプルプログラム

```
list
100 '１００行
110 '１１０行
120 ENTRY
130 PRINT "１２０行"
140 PRINT "１３０行"
Ok
edit .
140 PRINT "１３０行"

edit 110
110 '１１０行

edit entry
120 ENTRY
```

END
ステートメント

機 能	プログラムの終了を宣言する.
書 式	END
文 例	END

解 説

プログラムの実行を終了し，すべてのファイルをクローズしてコマンドレベルに戻ります.

END文はプログラムの実行を終了させたい所に置けばよく，また，いくつ置いてもかまいません. プログラムの最後のEND文は省略することができますが，この場合，ファイルのクローズは行われませんのでご注意ください.

サンプルプログラム

```
100  CLS
110 *LOOP
120  PRINT TIME$
130  A$=INKEY$
140   IF A$="s" OR A$="S" THEN END
150 GOTO *LOOP
```

```
15:35:51
15:35:51
15:35:51
15:35:51
15:35:51
15:35:51
15:35:51
15:35:51
15:35:51
15:35:51
15:35:51
15:35:51
Ok
```

ENVIRON

コマンド

機 能	環境エリアのパラメータを書き換える.
書 式	ENVIRON <パラメータ名＝文字列>
文 例	ENVIRON "PATH＝¥；¥BIN"
解 説	MS-DOS が BASIC 用に用意した環境エリアのパラメータを書き換えるために使用します．この文を使うことにより，SHELL 文で呼び出されるチャイルドプロセス用の環境エリアのパラメータを変えることができます（例えば DOS の PATH パラメータなど）.

" "の内部は常に大文字でなければなりません．また，MS-DOS で登録した文字数を越えて登録することはできません．

詳細については，BASIC ユーザーズガイドおよび MS-DOS ユーザーズガイドをご覧ください．

参 照　ENVIRON$，SHELL

サンプルプログラム

```
ENVIRON "PATH=A:¥USR"
Ok
print ENVIRON$("PATH")
A:¥USR
Ok
```

ENVIRON$

関数

機 能　環境エリアのパラメータを返す.

書 式　ENVIRON$（<パラメータ名>）

ENVIRON$（<パラメータ番号>）

文 例　A$=ENVIRON$（"PATH"）

解 説　<パラメータ名>で指定されたパラメータを，MS-DOS が BASIC 用に用意した環境エリアから検索し，そのパラメータに割り付けられている文字列を返します.

<パラメータ番号> が指定されたときは，その番号に位置するパラメータの割付け状態を返します.

詳細については，BASIC ユーザーズガイドおよび MS-DOS ユーザーズガイドをご覧ください.

参 照　ENVIRON，SHELL

サンプルプログラム

```
ENVIRON "PATH=A:¥USR"
Ok
print ENVIRON$("PATH")
A:¥USR
Ok
```

EOF

関数

機能 ファイルの終了状態を返す.

書式 EOF (〈ファイル番号〉)

文例 IF EOF (1) THEN CLOSE1 : END

解説 〈ファイル番号〉で指定されたファイルが終わりに達したかどうかを調べる関数です. 関数の値として終りに達していれば真 (−1), そうでなければ偽 (0) を返します. 〈ファイル番号〉で指定されたファイルは入力モードでオープンされていなければなりません.

指定されたファイルがRS-232Cコミュニケーション回線(COM1:)であった場合には, EOF関数は, 1文字入力があるまで待ち, その文字が "END OF FILE" コード (&H1A) であれば真を, そうでなければ偽を返します.

サンプルプログラム

```
100 '
110 OPEN "DATA5" FOR INPUT AS #1
120 IF EOF(1) THEN 160
130  INPUT #1,A
140  PRINT A
150 GOTO 120
160 CLOSE
```

```
run
 1
 2
 3
 4
 5
 6
 7
 8
 9
```

ERASE
ステートメント

機能　配列変数を消去する.

書式　ERASE <配列変数名>〔, <配列変数名>・・・〕

文例　ERASE C, D

解説　指定した配列変数を消去します. 以後, 同一変数名の配列変数を DIM 文で宣言することもできます. もし, ERASE 文を実行せずに同一変数名で宣言すると, "Duplicate Definition"エラーが起こります.

配列変数が消去されると, その配列変数名に割り当てられていたメモリエリアを, 他の目的に使用することができます.

参照　DIM

サンプルプログラム

```
100 DIM A(100),B(100)
110 ERASE A
120 DIM A(200)
130 DIM B(200) :' ｾﾝｹﾞﾝ ﾃﾞｷﾅｲ!!
```

```
run
Duplicate Definition in 130
Ok
```

ERDEV/ERDEV$

システム変数

機能	デバイスエラーの値，およびデバイスエラーを起こしたデバイス名を返す．

書式

ERDEV

ERDEV$

文例

D=ERDEV AND &HFF

D$=ERDEV$

解説

ERDEV は，デバイスエラーの値を得るためのシステム変数です．

キャラクタデバイスが何らかのエラーを起こした場合，この変数の下位 8 ビットには DOS の割込みタイプ 24H エラーコードが，上位 8 ビットにはデバイスヘッダブロックから得られるデバイスヘッダステータスワード中のビット15〜13がセットされます．

ERDEV$は，エラーを起こしたキャラクタデバイスの名前（8 文字）を返します．エラーを起こしたデバイスがキャラクタデバイスでない場合には，2 文字のデバイス名（A：，B：など）を返します．

割込みタイプ 24H エラーコード，デバイスヘッダステータスワード，キャラクタデバイスなどの詳細については MS-DOS プログラマーズリファレンスマニュアルをご覧ください．

参照

ON ERROR GOTO

サンプルプログラム

```
100 ON ERROR GOTO *ERRORDEVICE
110 PRINT "ディスクドライブ にディスクを入れてください。":GO
SUB *OK
120 PRINT "FILES":FILES
130 PRINT "ディスクを抜いてください。":GOSUB *OK
140 PRINT "FILES":FILES
150 END
160 *OK
170 PRINT "準備ができたらなにかＫＥＹを押してください。"
180 IF INKEY$="" THEN 180
190 RETURN
200 *ERRORDEVICE
220 IF ERDEV=2 THEN PRINT "装置";CDBL$(ERDEV$);"が異状です。
"
230 ON ERROR GOTO 0
240 END
```

```
run
ディスクドライブ にディスクを入れてください。
準備ができたらなにかKEYを押してください。
FILES
A:¥
IO      .SYS     MSDOS   .SYS     COMMAND .COM     PRINT   .COM
CHKDSK  .COM     RECOVER .COM     SYS     .COM     EDLIN   .COM
MORE    .COM     DISKCOPY.COM     DEBUG   .COM     SETUP   .COM
LOGOFF  .COM     HDFORM  .COM     HRDDRV  .SYS     GWBASIC .EXE
AUTOEXEC.BAT     BOLD    .SYS     ITALIC  .SYS     CONFIG  .SYS
AUTOEXEC.BAK
 131072 Bytes free
ディスクを抜いてください。
準備ができたらなにかKEYを押してください。
FILES
装置A:が異状です。
Disk not Ready in 140
Ok
```

ERL/ERR

システム変数

機 能

発生したエラーのエラーコード，およびエラーの発生した行番号を返す．

書 式

ERL

ERR

文 例

L=ERL

E=ERR

解 説

エラーが発生した時点で，システム変数 ERR はエラーコードを，ERL はエラーの発生した行番号を持っています．エラーがダイレクトモードの実行によって生じた場合，ERL は行番号として65535を持ちます．

一般に，ERR 及び ERL は ON ERROR GOTO 文によって指定したエラー処理ルーチンにおいて，処理の流れを制御するために使われます．

なお，ERL を IF 文中で使用する場合，ERL は，必ず算術演算子の左辺におくようにします．

IF ERL=30 THEN・・・

こうすることにより，RENUM コマンドを実行したとき，右辺の行番号もリナンバすることができます．

参 照

ON ERROR GOTO

サンプルプログラム

```
100 ' -- X ノ Y ジョウヲ モトメル
110  ON ERROR GOTO *ERRORTRP
120 *START
130  INPUT "X,Y";X,Y
140  ANS=X^Y
150 PRINT X;"ノ";Y;"ジョウ ハ";ANS;"デス"
160  GOTO *START
170 END
180 '-- ERROR ROUTINE --
190 *ERRORTRP
200  IF ERR=11 AND ERL=140 THEN PRINT "??? テイギサレマセン":RESU
ME 170
210  IF ERR=6 AND ERL=140 THEN PRINT "オオキスギマス":RESUME 170
220  ON ERROR GOTO 0
```

```
run
X,Y? 2,3
 2 ﾉ 3 ｼﾞｮｳ ﾊ 8 ﾃﾞｽ
X,Y? 10,4
 10 ﾉ 4 ｼﾞｮｳ ﾊ 10000 ﾃﾞｽ
X,Y? -1,3
-1 ﾉ 3 ｼﾞｮｳ ﾊ-1 ﾃﾞｽ
X,Y? 0,-1
??? ﾃｲｷﾞｻﾚﾏｾﾝ
Ok
```

ERROR

ステートメント

機　能　エラーが発生した状態にする．

書　式　ERROR 〈エラーコード〉

文　例　ERROR 200

解　説　〈エラーコード〉には0から255までの範囲の数を指定します．この値が，すでにBASICでエラーコードとして使われていれば，ERROR文はそのエラーを実際に発生させ，対応するエラーメッセージを表示します．

もしON ERROR GOTO文によってエラー処理ルーチンが用意されていれば，エラーメッセージは表示されずに，そのルーチンが実行されます．

なお，〈エラーコード〉がBASICのエラーコードとして未定義である場合には，それらの値をユーザーが使うことができます．即ち，未定義の値をERROR文で指定することによってエラーを人為的に発生させてやれば，エラー処理ルーチン内で，その値をもとに，対応するエラー処理を行うことができます．

参　照　ERL／ERR，ON ERROR GOTO，付録　エラーメッセージ

サンプルプログラム

```
100  ON ERROR GOTO *ERMESSAGE
110 *NYURYOKU
120  INPUT "3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｾﾖ:",N
130   IF FIX(N)<>N THEN ERROR 251
140   IF N<100 OR N>999 THEN ERROR 250
150  PRINT USING "<###>";N
160 END
170 *ERMESSAGE
180  IF ERR=250 THEN PRINT " 3 ｹﾀ ﾄﾊ 100 ｶﾗ 999 !!!"
190  IF ERR=251 THEN PRINT " ｼﾞｯｽｳ ﾊ ﾆｭｳﾘｮｸ ﾃﾞｷﾏｾﾝ "
200 PRINT
210 RESUME *NYURYOKU
```

```
run
3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｾﾖ:56464
 3 ｹﾀ ﾄﾊ 100 ｶﾗ 999 !!!

3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｾﾖ:65.5
 ｼﾞｯｽｳ ﾊ ﾆｭｳﾘｮｸ ﾃﾞｷﾏｾﾝ

3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｾﾖ:564
<564>
Ok
```

EXP

関数

機能　eに対する指数関数の値を返す.

書式　EXP (〈数式〉)

文例　E=EXP (1)

解説　自然対数の底eの〈数式〉乗を返します.

〈数式〉にはどんな型の数値を指定しても,EXP関数は倍精度の値を返します.

サンプルプログラム

```
100 '   EXP ｶﾗ SINH(ﾊｲﾊﾟﾌﾞﾘｯｸ ｻｲﾝ)
110 '          COSH(ﾊｲﾊﾟﾌﾞﾘｯｸ ｺｻｲﾝ) ｦ ﾓﾄﾒﾙ
120 INPUT D
130 HS=(EXP(D)-EXP(-D))/2:HC=(EXP(D)+EXP(-D))/2
140 PRINT "SINH";USING"###";D;:PRINT " = ";HS
150 PRINT "COSH";USING"###";D;:PRINT " = ";HC,
160 END
```

```
run
? 2
SINH  2 =  3.626860407847
COSH  2 =  3.7621956910836
Ok
```

FIELD
ステートメント

機能　ランダムファイルバッファに変数領域を割り当てる．

書式　FIELD〔#〕〈ファイル番号〉，〈フィールド幅〉AS〈文字変数〉
〔，〈フィールド幅〉AS〈文字変数〉・・・〕

文例　FIELD #1，64 AS A$，32 AS B$，32 AS C$

解説　ファイルバッファをランダムファイルバッファとして使用するために，ファイルバッファの中に文字変数の領域を割り当てます．

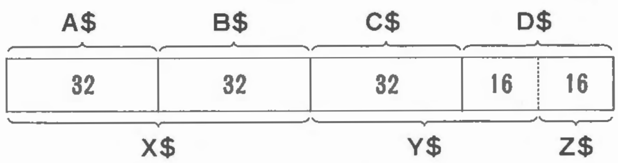
〈ファイル番号〉は，OPEN 文によってファイルをオープンしたときに指定した番号です．

〈フィールド幅〉は〈文字変数〉に割り当てる文字数です．１つのバッファには，文字数の合計がファイルバッファサイズ以下ならば，いくつの文字数を割り当ててもかまいません．

また，１つのバッファに対して，異なった形式の FIELD 文を実行してもかまいません．例えば，

```
FIELD #1, 32 AS A$, 32 AS B$, 32 AS C$, 32 AS D$
FIELD #1, 64 AS X$, 48 AS Y$, 16 AS Z$
```

とすれば，ファイル番号１のバッファに対し次のようにフィールドが割り当てられます．



つまり，変数 A$と B$を合わせたフィールドと変数 X$のフィールドが重なり，同じように C$と D$の前半が Y$，D$の後半が Z$に重なっています．こうすることにより，データをまとめて書き込んでおいてあとから分けて読み出したり，その逆を行ったりすることが容易にできるわけです．

フィールドを細かく多数設定したい場合，プログロム行の１行中に書き切れないことがあります．このようなときには，上記の重複フィールドを利用して次のようにします．

```
FIELD #1, 2 AS A$, 2 AS B$, 2 AS C$, 2 AS D$, ・・・, 2 AS H$

FIELD #1, 16 AS DUM1$, 2 AS I$, 2 AS J$, ・・・, 2 AS P$

FIELD #1, 32 AS DUM2$, 2 AS Q$, 2 AS R$, ・・・, 2 AS X$
```

DUM1$，DUM2$はダミーのフィールド変数で，こうすることにより A$～X$にはそれぞれ２バイトのフィールドを割り当てることができます．

PUT 文，GET 文を使ってランダムファイルのアクセスを行う前には，必ず FIELD 文が実行されて

いなければなりません.

FIELD 文はランダムファイルバッファ中に実際にデータを書き込むわけではなく，領域の割振りをするだけです．バッファ中にデータを書き入れるには，LSET 文と RSET 文を使います．

なお，ファイルバッファサイズは，BASIC の起動時に／S：パラメータで設定した値となります．／S：パラメータ省略時のサイズは128バイトです．

注意：FIELD 文中の〈文字変数〉は一般の文字変数として使用（INPUT 文の文字変数，LET 文の左辺など）しないでください．もし使用すると，その変数は一般の文字変数領域に登録され，FIELD 文でのバッファの割り当てが無効となり，正しいファイルの入出力が行われなくなります．

参照　OPEN，GET，PUT，LSET／RSET

サンプルプログラム

```
100 OPEN "RSET.DAT" AS #1
110 FIELD #1,30 AS A$,30 AS B$,30 AS C$
120 '-- DUMMY$ ﾊ A$+B$+C$ ﾉﾀﾒﾉ ﾀﾞﾐｰ ﾃﾞｽ｡ --
130 FIELD #1,90 AS DUMMY$,2 AS D$,4 AS E$,8 AS F$
140 LSET A$="NEC PC-100"
150 LSET B$="personal"
160 LSET C$="Computer"
170 RSET D$=MKI$(100)
180 RSET E$=MKS$(123456!)
190 RSET F$=MKD$(1.2345678901235E-35)
200 PUT #1,1
210 GET #1,1
220 PRINT A$
230 PRINT B$
240 PRINT C$
250 D=CVI(D$)
260 E=CVS(E$)
270 F=CVD(F$)
280 PRINT D:PRINT E:PRINT F
290 END

run
NEC PC-100
personal
Computer
 100
 123456
 1.2345678901235E-35
Ok
```

FILES

コマンド

機 能	ディレクトリ中のファイルをリストする.
書 式	FILES〔<ファイルスペック>〕
文 例	FILES "B : ¥DATA¥TEST.＊"

解 説

<ファイルスペック> で指定されるファイルをリストします. <ファイルスペック> はドライブ名＋パス名＋ファイル名から成ります. ファイル名が省略されたときには, ドライブ名, パス名で指定されたディレクトリのすべてのファイルをリストします.

ドライブ名が省略されると, カレントドライブが仮定され, パス名が省略されるとカレントディレクトリが仮定されます.

ファイル名にはワイルドカードと呼ばれるクェスチョンマーク（？）とアスタリスク（＊）を使うことができます. クェスチョンマーク（？）はファイル名中の1文字の代わりをすることができ, アスタリスク（＊）はファイル名中のファイル名または拡張子の代わりをすることができます. 例えば,

FILES "B : C????. BAS"

は, Bドライブのカレントディレクトリ中に存在し, 拡張子. BASをもち, Cではじまる5文字のファイル名をもったファイルをすべてリストします.

FILES "ROCK. ＊"

は, カレントドライブのカレントディレクトリ中に存在していて, ファイル名がROCKのファイルをすべて（どんな拡張子がついているものでも）リストします.

参 照 LOAD, BLOAD, SAVE, BSAVE

サンプルプログラム

```
files
A:¥
TEST1   .BAS      TEST2                 TEST3   .BAS      TEST4   .BAS
TEST5   .BAS
 357376 Bytes free
Ok
```

FIX
関数

機　能	整数部を返す．
書　式	FIX（〈数式〉）
文　例	F＝FIX（B／3）
解　説	〈数式〉の値の小数点以下を取り去った値を返します． INT との違いはサンプルプログラムを見れば明らかです．
参　照	INT
サンプルプログラム	

```
100 '-- FIX & INT --
110 A=-2
120 FOR I= 1 TO 9
130 A=A+I*.3
140 PRINT USING "A=+##.##      ";A,:PRINT "FIX(A)=";FIX
(A),"INT(A)=";INT(A)
150 NEXT I

run
A= -1.70        FIX(A)=-1       INT(A)=-2
A= -1.10        FIX(A)=-1       INT(A)=-2
A= -0.20        FIX(A)= 0       INT(A)=-1
A= +1.00        FIX(A)= 1       INT(A)= 1
A= +2.50        FIX(A)= 2       INT(A)= 2
A= +4.30        FIX(A)= 4       INT(A)= 4
A= +6.40        FIX(A)= 6       INT(A)= 6
A= +8.80        FIX(A)= 8       INT(A)= 8
A=+11.50        FIX(A)= 11      INT(A)= 11
Ok
```

FOR～NEXT

ステートメント

機　能　FOR 文から NEXT 文までの一連のステートメントやコマンドを指定回数だけ繰り返して実行する.

書　式

FOR 〈変数名〉＝〈初値〉TO〈終値〉〔STEP〈増分〉〕

　∫

NEXT〔〔〈変数名〉〕〔,〈変数名〉〕〕

文　例

```
FOR J=0 TO 100 STEP 2
  ∫
NEXT J
```

解　説　FOR～NEXT ループ中のステートメントやコマンドを繰り返し実行します.

〈変数〉はループカウンタとして使われ，最初に初値に設定されます．そして，FOR 文以降から NEXT 文までが実行され，カウンタの値は STEP によって指定された〈増分〉だけ増減されます.

次にカウンタの値が〈終値〉と比べられ，カウンタの値が〈終値〉に達していなければ，FOR 文の次の文へ戻り，同じ処理が繰り返されます.

STEP 文が省略された場合，〈増分〉は＋1とみなされます．また，〈増分〉は，負の値をとることもできます.

次の場合には，FOR～NEXT ループは実行されずに，NEXT 文の次の文へ実行が移ります.

1)〈増分〉が正の値で，〈初値〉が〈終値〉より大きい場合.
2)〈増分〉が負の値で，〈初値〉が〈終値〉より小さい場合.
3)〈初値〉と〈終値〉が同じ値で〈増分〉が0の場合.

ただし，〈変数〉には〈初値〉が代入されています.

FOR～NEXT ループはネスト（入れ子）構造にすることができます．これは1つの FOR～NEXT ループのなかにもう一つの FOR～NEXT ループを置くことができるということで，この場合，それぞれの〈変数〉は別のものを使わなければなりません．また，1つの FOR～NEXT ループは完全に他の FOR～NEXT ループあるいは WHILE～WEND ループの内部になければなりません．つまり，次のようなループは認められません.

```
FOR I = 0 TO. . .  ─────┐
FOR J = 1 TO. . .  ───┐ │
                      │ │
                      │ │
ENXT I  ──────────────┼─┘
ENXT J  ──────────────┘
```

次のようなループは正しいものです。

```
FOR I = 0 TO . . .  ─────┐
FOR J = 1 TO . . .  ──┐  │
                      │  │
                      │  │
ENXT J  ──────────────┘  │
NEXT I  ─────────────────┘
```

最後の2行のNEXT文は，1つにまとめて，NEXT J，Iとすることもできます．この場合，〈変数名〉の並び順がやはりループのネスト関係に影響をおよぼしますので注意してください．

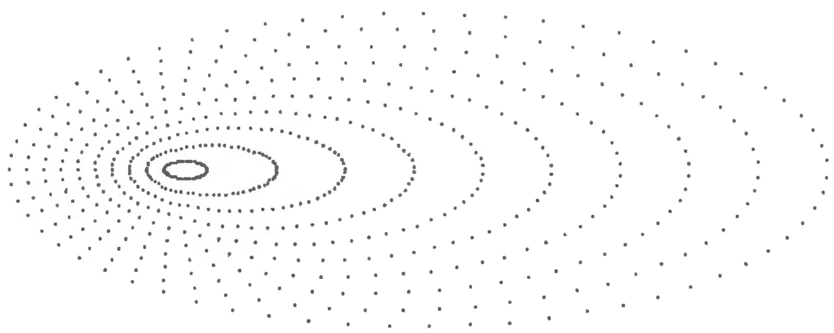
なお，FOR文の〈変数〉（ループカウンタ）とNEXT文の〈変数〉とは必ず1対1に対応していなければなりません．

また，FOR～NEXTループ中からGOTO文などでループの外に出てしまうようなプログラムは，正しい動作が保証されなくなりますので注意してください．

サンプルプログラム

```
100 SCREEN 1:CLS:LOCATE 1,1,0
110 FOR R=10 TO 200 STEP 20
120    CO=RND(1)*7+8
130  FOR I=0 TO 3.14 STEP .05
140   Y=SIN(I)*COS(I)
150   X=SIN(I)*SIN(I)
160   Y=Y*R+250
170   X=X*R*2.5+200-C
180   PSET (X,Y),CO
190  NEXT I
200  C=C+10
210 NEXT R
220 END

Ok
```



FRE
関数

機 能	メモリの未使用領域の大きさを返す.
書 式	FRE (<ダミー引数>)
文 例	PRINT FRE (0)

解 説

FRE 関数は BASIC の利用可能なメモリ領域のうちの，未使用領域の大きさを関数値として返します．<ダミー引数> には，定数，変数など，何を指定してもかまいません．

<ダミー引数> が文字列の時は，FRE 関数は，ストリング領域の整理のため〝ちり集め〟(Garbage collection) 処理を行いますので，関数が値を返すまでに時間がかかることがあります．

サンプルプログラム

```
100 REM '-- User Memory print out --
110 PRINT "メモリの未使用領域の大きさを返す。";FRE(0)
```

```
run
メモリの未使用領域の大きさを返す。 60228
Ok
```

GET

ステートメント

機能　ファイル中のデータをバッファに読み込む.

書式　GET〔#〕<ファイル番号>〔,<整数>〕

文例　GET 3, 5

解説　<ファイル番号>で指定されたファイル中のデータを,対応するバッファ中に読み込みます.GET は,指定されたファイルがディスクファイルかどうかで,その動作が異なります.

<ファイル番号>で指定されたファイルがディスクファイルである場合には,GET は,ランダムファイルからの読込みとして働きます.指定されたファイルは,ランダムファイルのモードでオープンされていなければなりません.

<整数>はファイルのレコード番号として解釈され,指定された1レコード分がバッファに読み込まれます.<整数>が省略された場合には,直前に GET,PUT が行われたレコードの次のレコードが読み込まれます.

GET を行った後で,INPUT #,LINE INPUT #等のシーケンシャル入力を行った場合には,バッファに入っているデータから入力が行われますので注意してください.

<ファイル番号>で指定されたファイルがディスクファイルでない場合には,GET は,指定されたファイルに対応する装置から指定した文字数をバッファ中に読み込みます.指定されたファイルは入力モードでオープンされていなければなりません.<整数>は,ファイルからバッファに読み込むバイト数を意味し,0からファイルバッファサイズまでの値で指定します.<整数>が省略されたとき,および0が指定されたときにはファイルバッファサイズ分のデータを読み込みます.

GET はバッファへの読込みを行うものです.変数に GET したデータを割り付けるには,FIELD 文を用います.

参照　OPEN, FIELD, PUT

サンプルプログラム

```
100 '--RANDOM DATA READ --
110 OPEN "DATA1" AS #1
120 FIELD #1,30 AS A$,20 AS B$,50 AS C$
130 *ENTRY
140 I=I+1
150 GET #1,I
160 IF EOF(1) THEN 220
170 PRINT
180 PRINT "NAME",A$
190 PRINT "TEL",B$
200 PRINT "ADDRESS",C$
210 GOTO *ENTRY
220 CLOSE
```

```
run

NAME          相沢
TEL           03-001-3418
ADDRESS       東京都　世田谷区　１－１－５

NAME          岡野
TEL           00-087-8734
ADDRESS       神奈川県　横浜市　１－２－３
Ok
```

GET@

ステートメント

機　能　画面上のグラフィックパターンを読み込む.

書　式　GET〔@〕(Sx1, Sy1) ― | (Sx2, Sy2) / STEP (x, y) | ,〈配列変数名〉〔(〈要素〉)〕

文　例　GET (100, 50) ― (130, 70), G%

解　説　画面上のスクリーンの座標上の2点 (Sx1, Sy1) と (Sx2, Sy2) を対角点とする四角形の領域を,〈配列変数名〉で指定された配列変数に読み込みます.

〈配列変数名〉は, グラフィックパターンを読み込むために用意する数値型の配列変数の名前です. なお, この配列変数に対して〈要素〉を指定することにより, グラフィックパターンを配列変数に読み込む際に, どの要素番号から格納し始めるかを指定することができます. 1つの配列変数に対して複数のグラフィックパターンを格納する場合に使用すると便利です. 省略した場合は配列の最初から格納します.

ただし, ユーザーはGET@命令に先だってこの配列変数を必要な大きさだけ確保するよう, DIM文によって宣言しておかなければなりません. この配列変数の大きさは, 読み込む領域の大きさ, 現在のスクリーンモード, 配列変数の型によって異なりますが, 1つの配列に1つのパターンしか読み込まない場合は, 次の計算式で求めることができます. 複数パターンの場合はそれぞれのパターンについて計算し, その分確保しなければなりません.

〈必要なバイト数〉=((〈横のピクセル数〉+7)¥8)*〈縦のピクセル数〉*M+4

ただし {白黒モードのとき――――M=1
　　　　 カラーモードのとき――M=4

〈添字の値〉=〈必要なバイト数〉¥N

Nは配列変数の型によって変わります.

　　整数型配列――――N=2
　　単精度型配列―――N=4
　　倍精度型配列―――N=8

ただしこの〈添字の値〉とは, 1次元配列で添字の下限が0のとき (OPTION BASE参照) の最小値です. 一般にGET@では1次元の配列変数を用います.
このGET@文はPUT@文と密接な関係にあり, 通常ペアで使用されます.
なお, 2点目 (Sx2, Sy2) の座標指定には相対座標による指定が許されています. また, GET@文の実行後は, LPは2点目に移ります.

注意: このGET@命令とPUT@命令は, スクリーン座標系 (文法編　ウィンドウとビューポート参照) に対して有効です. ワールド座標系に対して使用することはできませんのでご注意ください.

参照　PUT@, OPTION BASE

サンプルプログラム

```
100 CLS:DIM A%(715)
110 FOR I=64 TO 359:PRINT CHR$(I);:NEXT
120 GET@(0,0)-(50,50),A%
130 PUT@(0,0),A%,PRESET
140 PUT@(150,150),A%
150 PAINT(1,1),9,15
```

```
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[¥]^_`abcdefghijklmnopqrstuvwxyz{¦}~█_≠ヤ薦敵炎旧克
署葬灯楓利凹亨込匠｡｢｣､･ｦｧｨｩｪｫｬｭｮｯｰｱｲｳｴｵｶｷｸｹｺｻｼｽｾｿﾀﾁﾂﾃﾄﾅﾆﾇﾈﾉﾊﾋﾌﾍﾎﾏﾐﾑﾒﾓﾔﾕﾖﾗﾘﾙﾚﾛﾜﾝﾞ
ﾟ雪偵泌也磯危鰯趣栴汀備藪飲棋　、。，．・：；？！゛゜´｀¨＾￣＿ヽヾゝゞ〃仝
々〆〇ー―‐／＼～‖｜…‥‘’“”（）〔〕［］｛｝〈〉《》「」『』【】＋－±×÷
＝≠＜＞≦≧∞∴♂♀°′″℃￥＄¢£％＃＆＊＠§☆★○●◎◇◆□■△▲▽▼※〒→
Ok
LPRINT
Ok
```

```
@ABCDE
署葬灯
ﾟ雪偵
```

GOSUB

ステートメント

機能 サブルーチンのコールを行う.

書式 GOSUB <行番号>

文例 GOSUB GOOD

解説 <行番号>から始まるサブルーチンプログラムをコールします．コールされたサブルーチンプログラムは，そのサブルーチン内のRETURN文が実行されると，コールしたGOSUB文の次の文へリターンします．<行番号>にはラベルを用いることもできます．

サブルーチンプログラムとは，1つの他から独立したプログラムで，RETURN文で終わっているものをいいます．これらは，GOSUB文によって，いつでも何度でもコールすることができます．

1つのサブルーチンの中から他のサブルーチンをコールすることもでき，これをサブルーチンのネストと呼びます．このネストはメモリのスタック領域の容量が許す限り行うことができます．もし，スタック領域が足りなくなると〝Out of memory〟エラーが起こります．このような場合には，CLEAR文によってスタック領域を大きくとる必要があります．

サブルーチンは必ずGOSUB文によってコールされなければなりません．もし，単独で実行した場合，RETURN文に出会うと〝RETURN without GOSUB〟エラーが起こります．

参照 RETURN, CLEAR

サンプルプログラム

```
100 INPUT "好きな 数だけＢＥＥＰを鳴らします。何回ですか？",
B
110 T=B:GOSUB *BELL
120 FOR I= 1 TO 1000:NEXT
130 PRINT "前の２倍、鳴らします。
140 T=B*2:GOSUB *BELL
150 END
160 *BELL
170  FOR I= 1 TO T
180    PRINT I
190  BEEP
200       FOR J=1 TO 1000:NEXT J
210  NEXT I
220 RETURN
```

```
run
好きな 数だけＢＥＥＰを鳴らします。何回ですか？2
 1
 2
前の２倍、鳴らします。
 1
 2
 3
 4
Ok
```

GOTO/GO TO

ステートメント

機能　指定した行へジャンプする．

書式　GOTO 〈行番号〉

GO TO 〈行番号〉

文例　GOTO 500

解説　〈行番号〉で指定した行へジャンプします．〈行番号〉にはラベルを用いることもできます．

GOTO と GO TO は全く同機能です

注意：〝GO〞と〝TO〞の間のスペースは1個のみが許されます．2個以上のスペースを入れた場合，BASIC はそれを GOTO 文とは解釈しません．

サンプルプログラム

```
100 'goto
110 GOTO *START
120 PRINT "PC-100 ";:GOTO 160
130 PRINT "COMPUTER ";:GOTO 120
140 *FINE:END
150 PRINT "PERSONAL ";:GOTO 130
160 PRINT "BASIC ";:GOTO *FINE
170 *START
180 PRINT "NEC ";:GOTO 150

run
NEC PERSONAL COMPUTER PC-100 BASIC
Ok
```

HEX$

関数

機能　10進数を16進数に変換し，その文字列を返す．

書式　HEX＄(<数式>)

文例　A＄＝HEX＄(X)

解説　<数式>の値を16進数に変換して，その文字列を返す関数です．<数式>の値は，－32768から32767(または0から65535）までの範囲になければなりません．

これとは逆に16進数を10進数として扱うには，16進数の前に"&H"をつけます．詳しくは文法編　定数の項をご覧ください．

参照　VAL，OCT＄

サンプルプログラム

```
100 PRINT "     -0 -1 -2 -3 -4 -5 -6 -7 -8 -9"
110 FOR I=0 TO 9
120 PRINT STR$(I);"- ";
130  FOR J=0 TO 9
140   PRINT RIGHT$("0"+HEX$(I*10+J),2);" ";
150  NEXT J
160  PRINT
170 NEXT I
180 END
```

```
run
    -0 -1 -2 -3 -4 -5 -6 -7 -8 -9
 0- 00 01 02 03 04 05 06 07 08 09
 1- 0A 0B 0C 0D 0E 0F 10 11 12 13
 2- 14 15 16 17 18 19 1A 1B 1C 1D
 3- 1E 1F 20 21 22 23 24 25 26 27
 4- 28 29 2A 2B 2C 2D 2E 2F 30 31
 5- 32 33 34 35 36 37 38 39 3A 3B
 6- 3C 3D 3E 3F 40 41 42 43 44 45
 7- 46 47 48 49 4A 4B 4C 4D 4E 4F
 8- 50 51 52 53 54 55 56 57 58 59
 9- 5A 5B 5C 5D 5E 5F 60 61 62 63
Ok
```

IF～THEN～ELSE/IF～GOTO～ELSE

ステートメント

機　能　論理式の条件判断を行う.

書　式

```
IF 〈論理式〉 | THEN | 〈文〉     | | 〔ELSE | 〈文〉     | 〕
              |      | 〈行番号〉 | |        | 〈行番号〉 |
              | GOTO   〈行番号〉   |
```

文　例　IF A$= "y" THEN START ELSE 120

解　説　論理式の条件によってプログラムの実行を制御します．即ち〈論理式〉の値が真（0以外）ならば THEN 文または GOTO 文以降を実行し，偽(0)なら ELSE 文以降が実行されます．もし，ELSE 文以降が省略されている場合には，次の行が実行されます．

IF I> 8 THEN I=1 ELSE I=0

上の例では変数Iの値が8より大きければI＝1を実行し，変数Iの値が8以下ならばI＝0を実行します．THEN 文，ELSE 文のいずれかが実行されれば，次の行に実行が移ります．

IF D THEN 990 ELSE D=1 : GOTO 10

この例では，〈論理式〉の項が論理演算の形をしていませんが，正しく動作します．つまり，Dの値が0以外ならば990行へ飛び，Dの値が0ならばD＝1を実行後，10行へ飛びます．

IF～THEN～ELSE 文において，THEN 文や ELSE 文に続けて別の IF 文を置き，多重にすることができます．ただし，多重にできるのは1行に書ける範囲に限られます．この場合，プログラムの流れがわかりにくくなりやすいので，よく考えてプログラムを作成してください．

サンプルプログラム

```
100 '-- real or imaginary --
110 INPUT A
120 FLG=0
130  IF A<0 THEN A=ABS(A):FLG=1
140 PRINT SQR(A);
150  IF FLG THEN PRINT "i" ELSE PRINT
160 END

run
? 654
 25.573423705089
Ok
run
? -654
 25.573423705089 i
Ok
```

INKEY$

関数

機　能	キーが押されていればその文字を，キーが押されていなければヌルストリングを返す．
書　式	INKEY$
文　例	A$＝INKEY$

解　説

キーボードバッファが空であれば，INKEY$関数はヌルストリングを返します．キーボードバッファが空でない（つまりキー入力された）場合には，バッファの先頭から1文字取り出し，その文字を返します．

押されたキーは画面上には表示されません．また，プログラム中，INKEY$の書かれた箇所でプログラムが停止することはありません．したがって，キー入力待ちをするためには次のようにしなければなりません．

100 A$＝INKEY$: IF A$＝ " " THEN 100

こうすることにより，キー入力がない間は100行でループが形成されてキー入力待ちが実現されることになります．

なお，キー入力されたキャラクタをとりこむには，INKEY$関数のほか，INPUT 文，LINE INPUT 文，INPUT$関数，INPUT¥関数があります．

注意：[CTRL]+[C]，[STOP] キーは INKEY$関数によって拾うことはできません．

参　照　INPUT，LINE INPUT，INPUT$

サンプルプログラム

```
100   PRINT  "Enter any key to end ・・"
110 *LOOP
120   A$=INKEY$
130    IF A$="" THEN *LOOP
140    PRINT A$
150   IF A$="e" OR A$="E" THEN *EXIT ELSE *LOOP
160 *EXIT:END

run
Enter any key to end ・・
d
k
f
l
j
0
I
D
F
e
Ok
```

INP

関数

機 能　入力ポートから値を得る.

書 式　INP (<ポート番号>)

文 例　A=INP(15)

解 説　<ポート番号> で指定された入力ポートから8ビットのデータを読み取り，それを関数値とします. <ポート番号> として指定できる値の範囲は&H0～&HFF (0～255) までです. <ポート番号> と装置との対応はMS-DOS ユーザーズガイドを参照してください.

参 照　OUT

サンプルプログラム

```
100 'INP
110 'ＤＩＰスイッチを読むプログラム
120 '
130 A=INP(&H22)              'INPUT ADDRESS 22H
140 PRINT "ＤＩＰスイッチの設定状態"
150 PRINT
160 SW1=NOT A AND &H8      'SWITCH ON : LOW
170 SW2=NOT A AND &H4      'SWITCH OFF:HIGH
180 SW3=NOT A AND &H2
190 SW4=NOT A AND &H1
200 IF SW1<>0 THEN PRINT "ＳＷ１：ＯＮ      ＰＣ－１００では
必ずＯＦＦにする" ELSE PRINT "ＳＷ１：ＯＦＦ    ２Ｄ ｄｉｓ
ｋ用"
210 IF SW2<>0 THEN PRINT "ＳＷ２：ＯＮ      ドライブ１台用"
ELSE PRINT"ＳＷ２：ＯＦＦ    ドライブ２台用"
220 IF SW3<>0 THEN PRINT "ＳＷ３：ＯＮ      カラーディスプレ
イ用" ELSE PRINT "ＳＷ３：ＯＦＦ    モノクロディスプレイ用"
230 IF SW4<>0 THEN PRINT "ＳＷ４：ＯＮ      未使用" ELSE PRI
NT "ＳＷ４：ＯＦＦ    未使用"
240 END
run
ＤＩＰスイッチの設定状態

ＳＷ１：ＯＦＦ    ２Ｄ ｄｉｓｋ用
ＳＷ２：ＯＦＦ    ドライブ２台用
ＳＷ３：ＯＮ      カラーディスプレイ用
ＳＷ４：ＯＦＦ    未使用
Ok
```

INPUT

ステートメント

機 能　指定した変数へ入力する.

書 式　INPUT〔;〕〔<プロンプト文> | ; | 〕 <変数>〔, <変数>・・・〕
　　　　　　　　　　　　　　　　　　 | , |

文 例　INPUT "Date"; A$

解 説　INPUT 文が実行されると，プログラムはキーボードからの入力待ちの状態となります.

このとき，<プロンプト文> にセミコロン（;）が続いた場合には，<プロンプト文> の後にクエスチョンマーク（?）と1桁のスペースが画面に表示されますが，カンマ（,）が続いた場合には <プロンプト文> の後には何も出力されません.

INPUT 文の直後にセミコロン（;）があると，文字列の入力終了時に改行されません.

データはリターンキーを押すことによって入力され，変数に代入されます. <変数> をカンマ（,）で区切って複数個指定した場合には，入力するデータもカンマで区切って <変数> の数だけ入力しなければなりません. また，対応する <変数> とデータの型とは一致していなければなりません. もし，個数や型が違っていた場合には "? Redo from start" を出力して再び入力待ちとなります.

また，何も入力せずにリターンキーだけを押した場合は，数値変数ならば0が，文字変数ならばヌルストリングが入力されたとみなされます.

文字変数に文字列を代入する場合，カンマや文字列の前後の意味のある空白を入力したいときは，ダブルクォーテーション（"）で文字列を囲む必要があります. この場合は文字としてダブルクォーテーションを入力することはできません. これ以外の場合には，文字列をダブルクォーテーションで囲む必要はありません. しかし，その場合もダブルクォーテーションは文字列の最後の1文字にしか文字として使うことはできません.

サンプルプログラム

```
100 INPUT "名前   ";NA$
110 INPUT "性別   ";SEX$
120 INPUT "年齢  ";OLD
130 PRINT "名前   :  ";NA$
140 PRINT "性別   :  ";SEX$
150 PRINT "年齢   :";OLD;"才 "

run
名前  ? 古屋
性別  ? 女
年齢 ? 30
名前  : 古屋
性別  : 女
年齢  : 30 才
Ok
```

INPUT＃

ステートメント

機能　シーケンシャルファイルからデータを読み込む.

書式　INPUT＃〈ファイル番号〉,〈変数〉〔,〈変数〉・・・〕

文例　INPUT＃1, A, B

解説　データを読み込む対象がシーケンシャルファイルであることと，クエスチョンマーク(？)が出力されないことを除けば，INPUT 文とほぼ同機能です.

〈ファイル番号〉は，そのファイルが入力モードでオープンされたときに指定された番号です.

〈変数〉は，データを読み込むための変数です.

読み込むシーケンシャルファイルは，ディスク上のファイル，RS-232 C 回線からのデータ，キーボードからのデータなど，シーケンシャルな形をしているものなら何でもかまいません.

読み込まれるデータは，INPUT 文に対する応答として入力されるデータと同様の正しい形になっていなければなりません.

シーケンシャルファイルがディスク上のファイルである場合，INPUT＃文で読み込むデータは，ふつう PRINT＃文，または PRINT＃USING 文によって書き込まれたものです.

参照　OPEN, PRINT＃

サンプルプログラム

```
100 '-- seuential data read --
110  OPEN "stest.dat" FOR INPUT AS #1
120 *SINPUT
130  IF EOF(1) THEN END
140  INPUT #1,A$
150  PRINT A$
160  GOTO *SINPUT
170 END

run
CAT
DOG
HOUSE
SHEEP
MOUSE
Ok
```

INPUT$
関数

機 能　指定されたファイルまたはキーボードから指定したバイト数だけ文字を読み込む．

書 式　INPUT$（<バイト数>〔,〔#〕<ファイル番号>〕）

文 例　WORD$=INPUT$（6，#2）

解 説　<ファイル番号>によって指定されたファイルから<バイト数>分の文字列を読み込みます．<ファイル番号>が省略された場合には，キーボードから入力が行われますが，INPUT 文と異なり入力された文字は画面には表示されません．

INPUT$は指定された<バイト数>分だけ文字が入力されるまで待ち続けますが，既にバッファに入力済みのデータがある場合にはバッファ中から順に文字を拾ってきます．また INPUT$は，[STOP] キー，[CTRL]+[C] を除くすべての文字をそのまま読み込みますので，INPUT 文や LINE INPUT 文では入力することのできないラインフィードコードやキャリッジリターンコードなども入力することができます．

なお，漢字は 2 バイトキャラクタですので，漢字 n 字分を読み込みたいときには，<バイト数>に 2×n 分指定すればよいのですが，このとき語境界にじゅんぶん注意しないと 2 バイトキャラクタが 2 分されてしまい，読み込んだ文字を再現できなくなるおそれがあります．これを避けて，1 バイトキャラクタ，2 バイトキャラクタを問わず指定した文字数だけ文字を読み込みたいときには，INPUT¥関数を使います．

参 照　INPUT，LINE INPUT，INPUT¥

サンプルプログラム

```
100 '-- get key --
110 FOR I=1 TO 5
120  A$=INPUT$(I)
130  PRINT A$
140 NEXT
150 END
```

```
run
a
合
bcd
e意u
fghij
Ok
```

INPUT¥

関数

機能　指定されたファイルまたはキーボードから指定した文字数だけ文字を読み込む.

書式　INPUT¥（〈文字数〉〔,〔#〕〈ファイル番号〉〕）

文例　WORD$=INPUT¥（6）

解説　〈ファイル番号〉によって指定されたファイルから〈文字数〉分の文字列を読み込みます.〈ファイル番号〉が省略された場合には，キーボードから入力が行われますが，INPUT 文と異なり入力された文字は画面には表示されません.

INPUT¥は指定された〈文字数〉分だけ文字が入力されるまで待ち続けますが，既にバッファに入力済みのデータがある場合にはバッファ中から順に文字を拾ってきます.

また INPUT¥は，[STOP]キー，[CTRL]+[C] を除くすべての文字をそのまま読み込みますので，INPUT 文や LINE INPUT 文では入力することのできないラインフィードコードやキャリッジリターンコードなども入力することができます.

INPUT¥関数は，1バイトキャラクタも2バイトキャラクタも1文字として数えますので，必ず指定の文字数だけ文字を読み込むことができます．したがって，1バイトキャラクタと2バイトキャラクタの混在したデータなどを読み込むときに便利です.

これに対して，INPUT$関数は指定したバイト数だけデータを読み込みます.

参照　INPUT$, INPUT, LINE INPUT

サンプルプログラム

```
100 '-- get key --
110 FOR I=1 TO 5
120  A$=INPUT¥(I)
130  PRINT A$
140 NEXT
150 END
```

```
run
a
合意
bcd
ef売g
hijkl
Ok
```

INSTR

関数

機能　文字列の中から任意の文字列を捜してその文字の位置をバイト数で返す.

書式　INSTR (〔〈数式〉,〕〈文字列1〉,〈文字列2〉)

文例　J=INSTR (A$, "J")

解説　〈文字列1〉の中から〈文字列2〉を捜し，見つかれば発見した位置をバイト数で返します．もし見つからなければ0を返します．
〈数式〉は捜し始める位置をバイト数で指定します．もし省略した場合は〈文字列1〉の最初から捜し始めます．

〈文字列1〉,〈文字列2〉には漢字キャラクタを指定することもできますが，漢字キャラクタは2バイトコードですから，INSTR関数で返される値にはよく注意してください．例えば，

INSTR ("漢字", "字")

は，2ではなく，3を返します．

〈文字列1〉に2バイトキャラクタ（漢字）を指定し,〈文字列2〉に1バイトキャラクタ（ふつうの英数字）を指定したときは，INSTR関数が正しい値を返さないことがありますのでご注意ください．これは〈文字列1〉の漢字キャラクタの2バイト目の内部連続コードが〈文字列2〉の内部連続コードと一致してしまうことがあるためです．

なお,〈文字列2〉にヌルストリングを指定すると,〈数式〉と同じ値を返します．

参照　KPOS, KLEN, LEN

サンプルプログラム

```
100 '-- alphabet search --
110 A$="abcdefghijklmnopqrstuvwxyz"
120 PRINT "ｽｷﾅ ｱﾙﾌｧﾍﾞｯﾄ ﾉ KEY ｦ ｵｼﾃ ｸﾀﾞｻｲ"
130 ALP$=INPUT$(1)
140 ALP=ASC(ALP$)
150  IF ALP>64 AND ALP<91 THEN ALP=ALP+32
160 A=INSTR(A$,CHR$(ALP))
170  IF A=0 THEN PRINT ALP$;" ﾊ ｱﾙﾌｧﾍﾞｯﾄ ﾃﾞﾊｱﾘﾏｾﾝ!":END
180 PRINT USING "! ﾊ ## ﾊﾞﾝﾒﾉ ｱﾙﾌｧﾍﾞｯﾄﾃﾞｽ";ALP$,A
190 END

run
ｽｷﾅ ｱﾙﾌｧﾍﾞｯﾄ ﾉ KEY ｦ ｵｼﾃ ｸﾀﾞｻｲ
i ﾊ  9 ﾊﾞﾝﾒﾉ ｱﾙﾌｧﾍﾞｯﾄﾃﾞｽ
Ok
```

INT
関数

機能 小数点以下を切り捨てた整数値を返す.

書式 INT (<数式>)

文例 PRINT INT (1.123)

解説 <数式>の値を越えない最大の整数を返します.
FIX 関数との違いについては, FIX 関数の項を参照してください.

参照 FIX, CINT

サンプルプログラム

```
100   ' セイスウ ト ジッスウ ノ クベツ
110 A$="ハ ジッスウ デス"
120 B$="ハ セイスウ デス"
130 INPUT "スキナ カズ ヲ イレテ クダサイ";Z
140 R=Z=INT(Z):'コノ シキ ハ ロンリ エンザン デス
150 IF R=0 THEN PRINT Z;A$ ELSE PRINT Z;B$
160 END
```

```
run
スキナ カズ ヲ イレテ クダサイ? 6478
 6478 ハ セイスウ デス
Ok
run
スキナ カズ ヲ イレテ クダサイ? 564.56
 564.56 ハ ジッスウ デス
Ok
```

IOCTL
ステートメント

機 能	キャラクタデバイスドライバにコントロール文字列を送る．

書 式　IOCTL〔#〕<ファイル番号>，<文字列>

文 例　IOCTL #1，"00；SW132；GW"

解 説　IOCTL 文は，オープンされたキャラクタデバイスドライバに，コントロール文字列をバッファリングしないで送ることができます．

<ファイル番号>は，キャラクタデバイスドライバをオープンしたときに指定した番号です．

<文字列>は，キャラクタデバイスドライバに対する，コントロールコードを含む文字列コマンドです．この文字列コマンドはふつう英字＋数字の組合せから成る2，3文字のコマンドですが，これを複数個，セミコロンで区切って並べたコマンドの集まりである場合もあります．例えば，ユーザーが専用のドライバをLPT 2としてインストールし，これに対するページ内行数のセットコマンドが次のようであったとします．

PLn　　　（nは行数）

このとき，LPT 2ドライバをオープンしてページ内行数をセットするには，

```
OPEN "LPT2:" FOR OUTPUT AS #1
IOCTL #1, "PL66"
```

とします．これで，ページ内行数が66行のデバイスLPT 2がオープンされたことになります．

キャラクタデバイスは，キャラクタコード単位でデータを扱うデバイスのことで，ユーザーがインストールすることもできるものです．インストールの方法など，詳しいことについては，PC-100ユーザーズガイドをご覧ください．

サンプルプログラム

```
100 ' IOCTL
110 '
120  ' ユーザーが定義したキャラクタデバイスを開く
130  ' デバイス名を“ｃｈａｒｄｅｖ”とする
140 OPEN "¥dev¥chardev" AS 1
150  ' コントロール文字列“Ｗ８０”を送る
160 IOCTL #1,"W80"
170 CLOSE
180 END
```

IOCTL$

関数

機能	キャラクタデバイスドライバからコントロール文字列を読み出す.
書式	IOCTL$ ([#] <ファイル番号>)
文例	I$=IOCTL$ (2)

解説

IOCTL$関数は,オープンされたキャラクタデバイスドライバから,コントロール文字列を読み出すのに使います.

<ファイル番号>は,キャラクタデバイスドライバをオープンしたときに指定した番号です.

この関数は,IOCTL 文によってキャラクタデバイスに与えたコマンドが正しく受信されたか否かを調べたり,現在のキャラクタデバイスの状態を得たりするために使用します.例えば,次のようなプログラムがあったとします.

```
OPEN "FOO" AS #1
IOCTL #1, "RAW"
IF IOCTL$ (1)= "O" THEN CLOSE #1
```

これは,キャラクタデバイスドライバ "FOO" に,RAW データモードにせよ,というコマンドを送った結果,もしドライバ側から,モード切換え失敗との情報を返してきたら直ちにファイルをクローズする,というものです.

サンプルプログラム

```
100 ' IOCTL$
110 '
120  ' ユーザーが定義したキャラクタデバイスを開く
130  ' デバイス名を“chardev”とする
140 OPEN "¥dev¥chardev" AS 1
150  ' コントロール文字列“?W”を送る
160  '  “?W”は、デバイスから
170  ' あるステータスを返すことを要求する
180  ' コントロール文字列と、想定する
190 IOCTL #1,"?W"
200  ' デバイスから返事を受ける
210 A$=IOCTL$(#1)
220 CLOSE
230 END
```

JIS$
関数

機能 文字列を JIS コードにして返す.

書式 JIS$（<文字列>）

文例 A$＝JIS$（"漢"）

解説 <文字列>の最初の 2 バイトを16進表記 4 桁の JIS コード文字列に変換して返します.

<文字列>の内容がヌルである場合には, "Illegal function call" エラーを生じます.

なお, これとは逆に, CHR$関数などに JIS コードを直接使いたいときには, JIS コードの前に "&J" をつけます.

参照 付録　漢字コード表

サンプルプログラム

```
100 SCREEN 0,0,0,1
110 INPUT "何か漢字を入れてください。";KANZI$
120 PRINT " “"KANZI$;"” のＪＩＳコードは";JIS$(KANZI$);
"です。"
130 PRINT " “"KANZI$;"” の区点コードは";KTN$(KANZI$);"
です。"
140 END
```

```
何か漢字を入れてください。? 緑
“緑” のＪＩＳコードは4E50です。
“緑” の区点コードは4648です。
Ok
```

KEY

ステートメント

機能　1)ファンクションキーの内容の割付けを行う．
2)ユーザー定義キーの番号の割付けを行う．

書式　KEY 〈キー番号〉，〈文字列〉

文例　1)KEY 1，"LOAD"
2)KEY 15，"PC"

解説　〈キー番号〉は次のように指定します．

1～10：ファンクションキー[f･1]～[f･10]
15～20：ユーザー定義キー

KEY 文は，〈キー番号〉により，次のように異なる機能をもっています．
1）ファンクションキーに対して文字列を割り付ける（〈キー番号〉1～10）．
2）特定の文字列の並びに対して，キー割込み用のキー番号を割り付ける（〈キー番号〉15～20）．
次にそれぞれの場合について説明します．

1）ファンクションキーは10個あり，10個の〈文字列〉を記憶させておくことができます．各ファンクションキーは最大15文字までの文字列およびコントロール文字をプログラムでき，キーボードから入力できない文字は，CHR$関数をプラス記号で継いで使います．例えば，

KEY 1，"PALETTE"+CHR$（13）

とすれば，ファンクションキー1には"PALETTE"+"キャリッジリターンコード"が定義されたことになり，以後このキーを押すだけでP，A，L，E，T，T，E，リターン，と押したのと同様の効果を得ることができます．

2）ユーザー定義キーは，〈キー番号〉15～20の5個まで割り付けることができます．この場合，〈文字列〉には最大2文字まで指定可能です．例えば，

KEY 15，"AH"

とすれば，"AH"という文字列入力に対し，キー番号15が割り付けられたことになります．ファンクションキーがキー番号の決まったキーに文字列を定義するのに対し，ユーザー定義キーは文字列に対してキー番号を定義する点が異なります．ユーザー定義キーが割り付けられると，ON KEY（n）GOSUB 文と併用することにより，任意の文字列入力によるキー割込み処理を実現することができます．

キーの定義は，一度行うと，再度定義しなおすか，BASIC を再起動しない限り，その内容が変化することはありません．

参照 ON KEY (n) GOSUB

サンプルプログラム

```
10 KEY 1,"palette"
20 KEY 2,"console 1,25"
30 KEY 3,"files "
40 KEY 4,"list "
50 KEY 5,"run "+CHR$(13)
60 KEY 6,"color 15,0"
70 KEY 7,"delete "
80 KEY 8,"print "
90 KEY 9,"shell "
100 KEY 10,"cont"+CHR$(13)
110 NEW
```

```
key list
F1 palette
F2 console 1,25
F3 files
F4 list
F5 run
F6 color 15,0
F7 delete
F8 print
F9 shell
F10 cont
Ok
```

KEY LIST/ON/OFF

ステートメント

機能　ファンクションキーの内容をリストする．ファンクションキーの内容を画面最下行に表示させる，あるいは表示させない．

書式　KEY LIST
KEY ON
KEY OFF

文例　KEY LIST

解説　KEY LIST は，現在のファンクションキーの割付け状態がどうなっているかをリストします．各キーの初期割付け状態は次のようになっています．

f・1　load "
f・2　auto ␣
f・3　files ␣
f・4　list ␣
f・5　run CR
f・6　save "
f・7　delete ␣
f・8　print ␣
f・9　shell ␣
f・10　cont CR

KEY ON は，画面の最下行にファンクションキーの表示をさせるコマンドです．このファンクションキーの表示がされているときは，画面に表示可能な行数は1行分減ります．また，この表示はスクロールせず，CLS 文や PRINT CHR$（12）を実行しても消えることはありません．

KEY OFF は，上記のファンクションキーの表示をとりやめるコマンドです．

サンプルプログラム

```
key list
F1 load "
F2 auto
F3 files
F4 list
F5 run
```

```
F6 save "
F7 delete
F8 print
F9 shell
F10 cont
Ok
```

KEY(n) ON/OFF/STOP

ステートメント

機 能　キーによる割込みの許可，禁止，保留を設定する．

書 式　KEY（<キー番号>）ON
KEY（<キー番号>）OFF
KEY（<キー番号>）STOP

文 例　KEY(1)　ON

解 説　キーが押されたときに起こる割込みを許可するか(ON)，禁止するか(OFF)，保留するか(STOP)を設定します．
〈キー番号〉は，次のように指定します．

1～10：ファンクションキー［f･1］～［f･10］
11～14：カーソルキー（［↑］，［↓］，［←］，［→］）
15～20：ユーザー定義キー

ユーザー定義キー以外のキーは，KEY(n) ON／OFF／STOP 各文を実行するだけで，該当するキーの割込み設定を行うことができます．

ユーザー定義キーの割込み設定を行うときは，あらかじめ KEY 文で15～20のキー番号のうち任意の番号に文字列をセットしておく必要があります．例えば，

KEY 15, "AH"

と設定すれば，キーボードから"A"，"H"と続けて入力したときに，キー番号15のキーが押された，と解釈されるわけです．

KEY(n) ON 文は，キー割込みを許可します．このステートメント実行後は，指定された〈キー番号〉のキーを押すごとに割込みを発生し，ON KEY(n) GOSUB 文によって定義された処理ルーチンに分岐します．

KEY(n) OFF 文は，キー割込みを禁止します．このステートメント実行後は，キーを押しても処理ルーチンへの分岐は起こりません．

KEY(n) STOP 文は，キー割込みを保留します．このステートメント実行後は，キーを押しても押されたことを覚えているだけで，処理ルーチンへの分岐は起こりません．しかし，その後 KEY (n) ON ステートメントによって割込みが許可されると，先程のキーを押したことによって処理ルーチンに分岐します．

なお，割込み処理ルーチンに制御が移ると自動的に KEY(n) STOP（割込み保留）状態となりますので，割込み処理ルーチンの先頭で KEY(n) OFF 文あるいは KEY(n) STOP 文を実行する必要はありません．

参 照　ON KEY(n) GOSUB

サンプルプログラム

```
100 CLS
110 KEY 15," "
120 ON KEY(15) GOSUB HIZUKE
130 KEY(15) STOP
140 LOCATE 11,20,0:PRINT "只今の時刻は";TIME$;"です。"
150 LOCATE 13,10,0:PRINT "ｓｐａｃｅバーを押すと日付を
表示します。"
160 KEY(15) ON
170 GOTO 130
180 HIZUKE:KEY(15) OFF
190 LOCATE 11,20,0:PRINT "今日の日付は";DATE$;"です。"
200 LOCATE 13,10,0:PRINT "なにかｋｅｙを押すと元にもど
ります。      "
210 IF INKEY$="" THEN 210
220 CLS
230 KEY(15) ON:RETURN
```

只今の時刻は18:00:00です。

ｓｐａｃｅバーを押すと日付を表示します。

KILL

コマンド

機能　ディスクからファイルを削除する.

書式　KILL 〈ファイルスペック〉

文例　KILL "A : ¥DATA¥SAMPLE. ASC"

解説　〈ファイルスペック〉で指定したファイルをディスクから削除します.
〈ファイルスペック〉中のドライブ名, パス名が省略されたときには, それぞれカレントドライブ, カレントディレクトリが仮定されます.

また,〈ファイルスペック〉中のファイル名は, 拡張子(. BAS など) もふくめたフルネームで指定しなければなりません. この場合, アスタリスク (*), クェスチョンマーク (?) などのワイルドカードも使用することができます. ワイルドカードについては, FILES コマンドの項を参照してください.

なお, 削除するファイルはクローズ状態になっていなければなりません. もしオープン状態のファイルを削除しようとすると "File already open" エラーが起こります.

KILL コマンドはすべてのディスクファイル(プログラム, ランダムデータ, シーケンシャルデータファイル) について使用できます.

サンプルプログラム

```
100 OPEN "TEST2" FOR OUTPUT AS #1
110 PRINT "TEST2 ト イウ FILE ヲ ツクル"
120 PRINT #1."KILL statement test"
130 CLOSE
140 '
150 FILES:PRINT
160 PRINT "TEST2 ヲ サクジ゛ョ スル"
170 KILL "TEST"
180 FILES

run
TEST2 ト イウ FILE ヲ ツクル
A:¥
TEST1    .BAS      TEST2                TEST3    .BAS      TEST4    .BAS
TEST5    .BAS
 357376 Bytes free

TEST2 ヲ サクジ゛ョ スル
File not found in 170
Ok
```

KLEN
関数

機能　文字列の文字数を返す．

書式　KLEN（〈文字列〉）

文例　L＝KLEN（B$）

解説　〈文字列〉の文字数を返します．1バイトキャラクタ（連続コードが255以下の文字）も，2バイトキャラクタ（漢字など，連続コードが256以上の文字）も，1文字として数えます．

これに対し，LEN 関数は文字列のバイト数を返します．

参照　LEN，INSTR，KPOS

サンプルプログラム

```
100 K$="この漢字を含む文字列の長さは"
110 PRINT K$;
120 PRINT KLEN(K$);"です。"
130 END
```

```
run
この漢字を含む文字列の長さは 14 です。
Ok
```

KPOS
関数

機能 文字列の先頭からの位置をバイト数で返す.

書式 KPOS (<文字列>, <数式>)

文例 P=KPOS (A$, 5)

解説 <文字列> の <数式> 番目の文字の位置をバイト数で返します.

<数式> は, <文字列> の文字を先頭から数えたときの値で, 1バイトキャラクタも2バイトキャラクタも1文字に数えます. したがって,

KPOS ("漢字CHARACTER", 4)

は, 6を返します ("漢", "字" が2バイトキャラクタ).

参照 KLEN, LEN, INSTR

サンプルプログラム

```
10 A$="ＫＰＯＳのｻﾝﾌﾟﾙＬＩST"
20 PRINT A$
30 LOCATE 3,1:PRINT"I ﾓｼﾞﾒ         ﾊﾞｲﾄ"
40 FOR I=1 TO 13
50 PRINT I,KPOS(A$,I)
60 NEXT

run
ＫＰＯＳのｻﾝﾌﾟﾙＬＩST
I ﾓｼﾞﾒ        ﾊﾞｲﾄ
 1             1
 2             3
 3             5
 4             7
 5             9
 6             11
 7             12
 8             13
 9             15
 10            17
 11            19
 12            21
 13            22
Ok
```

KTN$

関数

機能 文字列を JIS 区点コードにして返す.

書式 KTN$（<文字列>）

文例 K$=KTN$（A$）

解説 〈文字列〉の最初の2バイトを16進表記4桁の JIS 区点コード文字列に変換して返します.

〈文字列〉の内容がヌルである場合には“Illegal function call”エラーを生じます.

なお，これとは逆に，CHR$関数などに JIS 区点コードを直接使いたいときには，JIS 区点コードの前に“&K”をつけます.

参照 付録　漢字コード表

サンプルプログラム

```
100 SCREEN 0,0,0,1
110 INPUT "何か漢字を入れてください。";KANZI$
120 PRINT " “"KANZI$;"” の区点コードは";KTN$(KANZI$);"
です。"
130 PRINT " “"KANZI$;"” のＪＩＳコードは";JIS$(KANZI$);
"です。"
140 END
```

```
何か漢字を入れてください。? 緑
“緑” の区点コードは4648です。
“緑” のＪＩＳコードは4E50です。
Ok
```

LCOPY

コマンド

機　能　ディスプレイ画面に表示されている文字，図形をプリンタにコピーする．

書　式　LCOPY

文　例　LCOPY

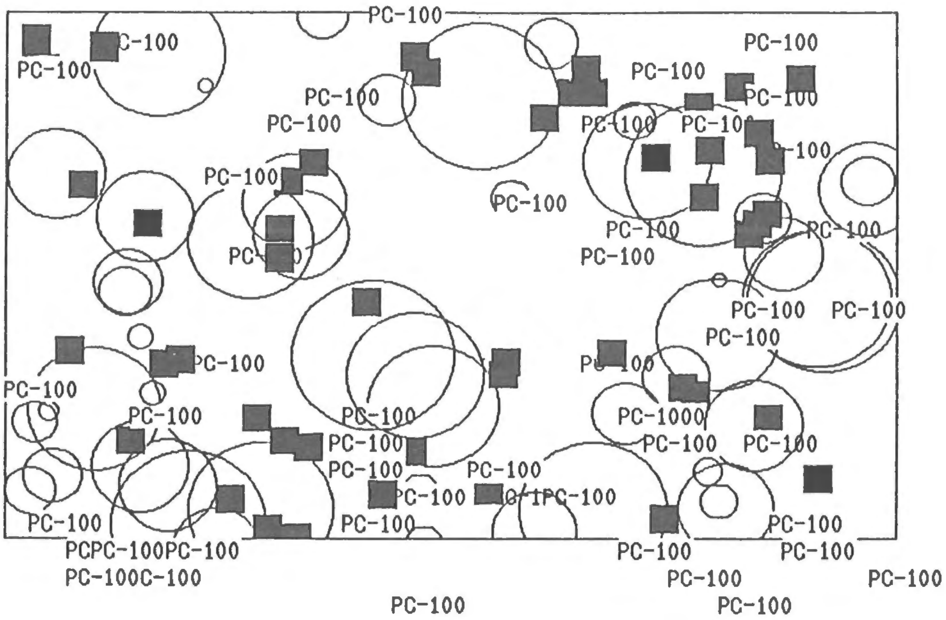
解　説　ディスプレイ画面に表示されている文字や図形をビットイメージで（画面上に表示されたとおりの形で）プリンタに出力します．

［COPY］キーを押すのとまったく同様の動作をします．

プリンタがプリンタドライバと一致していないとうまく動作しない場合があります．プリンタドライバのインストール方法は，MS-DOS ユーザーズガイド付録を参照してください．

サンプルプログラム

```
100 SCREEN 1,0,0,0:CLS:LOCATE 1,1,0
110 VIEW(1,9)-(638,400),,15
120 GOSUB *DISPLAY
140  LCOPY
160 COLOR 15:END
170 *DISPLAY
180  FOR J=1 TO 50
190  LOCATE RND*22+1,RND*70+1:COLOR RND*13+1:PRINT "PC-100"
200  LINE(RND*639,RND*400)-STEP(20,20),INT(RND*15+1),BF
210  CIRCLE(RND*639,RND*400),RND*60,INT(RND*7+1)
220 NEXT J:RETURN
```



LEFT$

関数

機能	文字列の左側から任意の長さの文字列を返す.
書式	LEFT$ (〈文字列〉, 〈数式〉)
文例	B$=LEFT$ (A$, 4)

解説

〈文字列〉の左側（先頭）から〈数式〉バイト分の長さの文字列を返します.
〈数式〉は0から255の範囲になければなりません.〈数式〉が,〈文字列〉の総バイト数以上の場合は,〈文字列〉のすべてを結果とします.〈数式〉が0ならば，ヌルストリングを結果とします.

〈文字列〉に2バイトキャラクタ（漢字など）を含んでいる場合,〈数式〉の値によっては2バイトキャラクタを2分してしまい, LEFT$関数は予期せぬ文字列を返すことがありますので語境界にはじゅんぶんご注意ください.

なお, LEFT$関数を使用しても，もとの〈文字列〉に変化を与えることはありません.

参照　RIGHT$, MID$（関数）, MID$（ステートメント）

サンプルプログラム

```
100 'ﾓｼﾞﾚﾂ(10) ﾉ ｶｲﾃﾝ ( くｰ )
110  INPUT A$
120  FOR I=1 TO 10
130   PRINT A$
140   B$=LEFT$(A$,1)
150   A$=MID$(A$,2)+B$
160  NEXT
170 END

run
? abcdefghijk
abcdefghijk
bcdefghijka
cdefghijkab
defghijkabc
efghijkabcd
fghijkabcde
ghijkabcdef
hijkabcdefg
ijkabcdefgh
jkabcdefghi
Ok
```

LEN
関数

機能	文字列の総バイト数を返す.
書式	LEN (<文字列>)
文例	PRINT LEN (A$)
解説	出力されない文字や空白も数えます. 出力されない文字とはキャラクタコードで1から31までの通常コントロールコードと呼ばれるものをいいます. 漢字などは2バイトキャラクタですから, 1文字を2バイト分に数えます. 1バイトキャラクタと2バイトキャラクタの混入した文字列の"文字数"を得たいときには, KLEN関数を用います.
参照	KLEN, KPOS

サンプルプログラム

```
100  INPUT "ｽｷﾅ ｺﾄﾊﾞｦ ｲﾚﾃ ｸﾀﾞｻｲ? ",A$
110  L=LEN(A$)
120  GOSUB *FLAME
130  PRINT "* ";A$;" *"
140  GOSUB *FLAME
150 END
160 *FLAME
170  FOR I=1 TO L+4
180   PRINT "*";
190  NEXT I
200  PRINT
210 RETURN

run
ｽｷﾅ ｺﾄﾊﾞｦ ｲﾚﾃ ｸﾀﾞｻｲ? I love you
**************
* I love you *
**************
Ok
```

LET
ステートメント

機能	変数に値を代入する.
書式	〔LET〕<変数名>＝<式>
文例	LET A＝10

解説 LET はなくてもかまいません. つまり<式>の値を<変数名>に代入するには, 等号だけでよいのです.

<式>の内容は, 数値, 文字列のいかなる型であってもかまいません. ただし, 文字型から数値型への代入またはその逆は許されません. 型の異なる数値変数への代入では左辺の型に変換されます.

サンプルプログラム

```
100 LET A=123
110 B=.21
120 LET C=A+B
125 D=A*B
130 PRINT "A   =";A
140 PRINT "B   =";B
150 PRINT "A+B =";C
155 PRINT "A*B =";D
160 END
```

```
run
A   = 123
B   = .21
A+B = 123.21
A*B = 25.83
Ok
```

LINE
ステートメント

機　能　指定された2点間に直線を引く．

書　式

LINE〔 (Wx1，Wy1) ／ STEP (x1，y1) 〕－ (Wx2，Wy2) ／ STEP (x2，y2) 〔，〈パレット番号〉〕〔， B ／ BF 〕

〔，〈ラインスタイル〉〕

文　例　LINE (100，100)－(135，100)，7，，&HF99F

解　説　ワールド座標系の2点（Wx1，Wy1）と（Wx2，Wy2）を結ぶ直線を描きます．（Wx1，Wy1）が省略された場合，LPの値を採用します．STEPを付けると相対座標による指定になります(文法編座標の指定の仕方参照)．

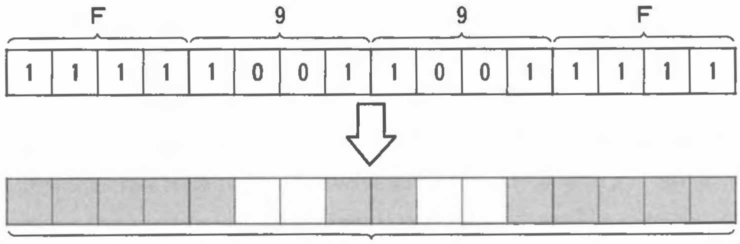
〈パレット番号〉には引く線の色を指定します．もしこれが省略された場合は，現在COLOR文によって設定されている，ディスプレイ画面のフォアグラウンドカラーが採用されることになります．次のパラメータには，BかBFのどちらかを指定することができます．BはBoxの意味で，(Wx1，Wy1）と（Wx2，Wy2）の2点を対角点とする四角形の箱を描き，BFはBox Fillの意味でその四角形を塗りつぶします．ただし〈ラインスタイル〉を指定した場合は，BFオプションを付けることはできません．

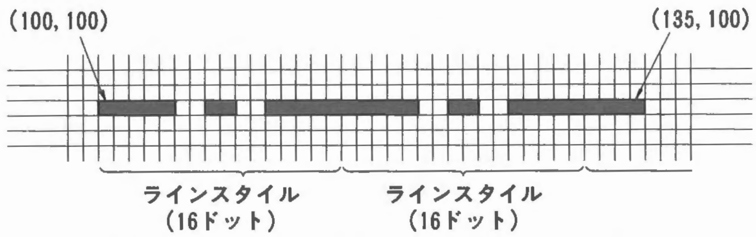
〈ラインスタイル〉は，描く線の形態を設定するオプションパラメータであり，&H0から&HFFFFの値をとることができます．このパラメータの値とラインの型態との関係は次の図に示す通りで，パラメータの16ビットと画面上の16点が1対1に対応しており，〝1″の立っているビットに対応する点が表示され，〝0″のビットに対応する点は表示されません（ここで，表示されない，とは，バックグラウンドカラーで描かれる，ということを意味します)．このパラメータは別に16進数でなくても整数型の数値であれば何でもよいのですが，16ビットに対応させて考える場合，16進数の方がわかりやすく一般的です．

線の長さを16点以上描いた場合には，このラインスタイルが画面上の16点ごとに繰り返されることになります．

次の図は，文例を実行した場合のものですが，水平座標の100から135までの36ドットの間に，1点鎖線のラインスタイルが繰り返されているのがわかります．
このようにラインスタイルを指定すると，16点単位で線の形態を決めることができますから，折れ線グラフや図面の作成の際，点線や1点鎖線など簡単に用いることができます．〈ラインスタイル〉が省略された場合は，ノーマルな直線を描きます．

ラインスタイル(&HF99F)





LINE(100, 100)－(135, 100), 7, , &HF99Fを実行した時

なお，LINE 文を実行した場合，LP は（Wx2，Wy2）に移動します．

サンプルプログラム

```
100 SCREEN 1,0,0,0:CLS
110 LOCATE 20,1,0
120 LINE(0,500)-(500,0),5
130 LINE(10,10)-(300,340),14,B
140 LINE(300,200)-(600,500),11,BF
150 LINE(0,150)-STEP(720,0),14,,&HAAAA
160 LINE(0,165)-STEP(720,0),13,,&HCCCC
170 LINE(0,180)-STEP(720,0),12,,&H8888
180 LINE(0,195)-STEP(720,0),10,,&HF99F
```

LINE INPUT

ステートメント

機　能	1行分のデータを文字変数に入力する.
書　式	LINE INPUT〔;〕〔<プロンプト文>;〕<文字変数>
文　例	LINE INPUT "NAME"; NA$

解　説

<プロンプト文>は，入力を受ける前にディスプレイ画面に表示される文字列です．クエスチョンマーク（?）は表示されません．プロンプト以降キャリッジリターンまでに入力されたすべてのデータ（ただし，255文字以内）が<文字変数>に代入されます．

LINE INPUT 文の直後にセミコロン（;）があると，文字列の入力終了時に改行されません．

LINE INPUT 文は，INPUT 文と違い，カンマ（,）によってデータが区切られることはありません．また，コロン（:），セミコロン（;），スラッシュ（/）などの各種の記号類も入力することができます．

LINE INPUT 文の実行中，CTRL+C または STOP キーを入力すると，LINE INPUT 文の実行は中断され，<文字変数>にデータは代入されません．この後 CONT コマンドを入力すると，再びその LINE INPUT 文から実行が再開されます．

注意：LINE と INPUT の間には，必ず1個スペースを入れなければなりません．

参　照　LINE INPUT #，INPUT，INPUT$，INPUT¥

サンプルプログラム

```
100 INPUT "INPUT : ";A$
110 LINE INPUT "LINE INPUT : ?";B$
120 PRINT:PRINT "INPUT : ",A$
130 PRINT "LINE INPUT : ",B$
```

```
run
INPUT : ? abcd,efgh,1234
?Redo from start
INPUT : ? abcd
LINE INPUT : ?abcd,efgh,1234,"HELLO"

INPUT :         abcd
LINE INPUT :    abcd,efgh,1234,"HELLO"
Ok
```

LINE INPUT#
ステートメント

機能　シーケンシャルファイルから１行分のデータを文字変数に読み込む．

書式　LINE INPUT # 〈ファイル番号〉，〈文字変数〉

文例　LINE INPUT #1，A$

解説　LINE INPUT #文は，シーケンシャルファイルから，キャリッジリターンコード（CHR$（13））＋ラインフィードコード（CHR$（10））で区切られている１行分（254バイト以内）のデータを〈文字変数〉へ読み込みます．

〈ファイル番号〉は，そのファイルが入力モードとしてオープンされたときに指定された番号です．

〈文字変数〉は，１行全体が代入される文字変数名です．

PRINT #文で作成したシーケンシャルファイルには，PRINT #文を実行するごとにデータ（数値，文字列，カンマなどのデータ区切り記号など）がキャリッジリターンコード＋ラインフィードコードで区切られて書き込まれています．

また，SAVE 文によってアスキー形式でセーブされたプログラムファイルは，１行ごとにキャリッジリターンコード＋ラインフィードコードで区切られて書き出されています．

LINE　INPUT #文は，キャリッジリターンコード（CHR$（13））＋ラインフィードコード（CHR$（10））の組合せ以外は区切り情報として見ないため，これらのファイルを行単位で入力したいときにとくに有用です．

参照　INPUT #

サンプルプログラム

```
100 A$="abcde,fghij,klmno,pqrst"
110 PRINT "OUTPUT DATA : ";A$
120 PRINT
130 OPEN "test7" FOR OUTPUT AS #1
140 PRINT#1,A$
150 CLOSE
160 OPEN "test7" FOR INPUT AS #1
170 INPUT#1,B$
180 PRINT"INPUT# : "; B$
190 CLOSE
200 OPEN "test7" FOR INPUT AS #1
210 LINE INPUT#1,C$
220 PRINT "LINE INPUT# : ";C$
230 CLOSE
```

```
run
OUTPUT DATA : abcde,fghij,klmno,pqrst

INPUT# : abcde
LINE INPUT# : abcde,fghij,klmno,pqrst
Ok
```

LIST/LLIST
コマンド

機　能　メモリ内にあるプログラムの全部または一部をリストアウトする.

書　式　LIST〔<行番号1>〕〔−〔<行番号2>〕〕
LLIST〔<行番号1>〕〔−〔<行番号2>〕〕

文　例　LIST 100−200
LIST .

解　説　LIST コマンドはプログラムの内容をディスプレイ画面に，LLIST コマンドはプログラムの内容をプリンタに出力（リストアウト）します.

<行番号1> のみを指定すると，その行だけがリストアウトされます.

<行番号1>−を指定すると，その行以降のプログラムがすべてリストアウトされます.

−<行番号2> を指定すると，プログラムの先頭からその行までがリストアウトされます.

<行番号1>−<行番号2> を指定すると，その行番号の範囲のプログラムがリストアウトされます.

<行番号> に現在の行を表すピリオド（.）を指定することもできます.

リストアウトの途中で PAUSE キーまたは CTRL + S を押すとリストアウトを一時中断することができます．このとき何かキーを押すとリストアウトが再開されます.

また，リストアウトの途中で STOP キーまたは CTRL + C を押すとリストアウトを中止することができます.

注意：LIST あるいは LLIST の後には，必ずスペースを入れてください.

サンプルプログラム

```
list
100 PRINT "１００行"
110 PRINT "１１０行"
120 *ENTRY
130 PRINT "１３０行"
Ok
list -*entry
100 PRINT "１００行"
110 PRINT "１１０行"
120 *ENTRY
Ok
```

```
list 110
110 PRINT "１１０行"
Ok
list 120-
120 *ENTRY
130 PRINT "１３０行"
Ok
list 110-120
110 PRINT "１１０行"
120 *ENTRY
Ok
list .
120 *ENTRY
Ok
```

LOAD
コマンド

機能	プログラムをメモリにロードする.
書式	LOAD <ファイルスペック> 〔, R〕
文例	LOAD "B : ¥BIN¥TEST"

解説

<ファイルスペック> で指定されたプログラムファイルをメモリ上にロードします.

ロードするプログラムファイルは, SAVE 文によってディスク上にセーブされた, 拡張子に". BAS"をもつファイルでなくてはなりません. しかし, LOAD 文では, <ファイルスペック>中のファイル名に拡張子". BAS"を指定する必要はありません. 例えば, ディスク B : 上のプログラムファイル"SEND. BAS"をロードするには, 次のいずれの形式でもかまいません.

```
LOAD "B : SEND"
LOAD "B : SEND. BAS"
```

<ファイルスペック> 中のドライブ名, パス名が省略されたときには, それぞれカレントドライブ, カレントディレクトリが仮定されます.

LOAD 文によってプログラムをロードすると, そのとき開かれているファイルはすべて閉じられ, 変数はもっている値をすべて失います.

R オプションをつけた場合は, そのとき開かれているファイルは閉じられることなく, ロードされたプログラムはただちに実行を開始します.

参照 SAVE, MERGE, RUN

サンプルプログラム

```
load "test"
load "b:¥demo¥test3"
load "test",r
```

LOC
関数

機能　ファイル中での論理的な現在位置を返す.

書式　LOC (<ファイル番号>)

文例　LAST=LOC (2)

解説　<ファイル番号>で指定されたファイルがランダムファイルであった場合には，LOC関数は，そのランダムファイルに対して最後にアクセス（GET／PUT）されたレコード番号を返します.

<ファイル番号>で指定されたファイルが，シーケンシャルファイルであった場合には，そのファイルがオープンされてから，読み書きされたレコード数を返します．この場合，ファイルバッファサイズ分をもって1レコード分とみなします．ファイルバッファサイズはBASICの起動時に／S：パラメータによって設定された値で，省略時には128バイトです.

指定されたファイルがRS-232Cコミュニケーションファイルであった場合には，LOC関数は，通信バッファ中にたまっているバイト数を値として返します.

なお，コミュニケーションファイル用の通信バッファサイズは，BASICの起動時に／C：パラメータによって設定された値となり，省略時には約256バイトです.

指定されたファイルがキーボードであった場合には，LOC関数はキーボードバッファにたまっている文字数を返します.

参照　LOF

サンプルプログラム

```
100 OPEN "data6" AS #1
110 FIELD #1,20AS A$,8 AS B$
120 INPUT "ﾅﾏｴ";NA$:INPUT "ｷｭｳﾖ";KY#
130 LSET A$=NA$:RSET B$=MKD$(KY#)
135 PUT #1
140 PRINT NA$;" ｻﾝﾉ data ﾊ";LOC(1);"ﾊﾞﾝﾃﾞｽ｡"
150 INPUT "continue?(y/n)";C$
160 IF C$="y" OR C$="Y" THEN PRINT :GOTO 120
170 IF C$<>"n" AND C$<>"N" THEN 150 ELSE END
```

```
run
ﾅﾏｴ? OKANO
ｷｭｳﾖ? 50000
OKANO ｻﾝﾉ data ﾊ 1 ﾊﾞﾝﾃﾞｽ｡
continue?(y/n)? Y

ﾅﾏｴ? AIZAWA
ｷｭｳﾖ? 150000
AIZAWA ｻﾝﾉ data ﾊ 2 ﾊﾞﾝﾃﾞｽ｡
continue?(y/n)? N
Ok
```

LOCATE

ステートメント

機能 テキスト画面のカーソルの位置を指定する．

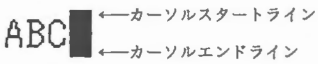
書式 LOCATE〔Y〕〔, X〕〔,〈カーソルスイッチ〉〕〔,〈カーソルスタートライン〉〕〔,〈カーソルエンドライン〉〕

文例 LOCATE 10, 10

解説 カーソルをディスプレイ画面のキャラクタ座標(Y, X)の位置へ移動します．Yは垂直座標を表し，省略された場合は現在カーソルがセットされている行とみなされます．Xは水平座標を表し，省略した場合は1とみなされます．これらは共に画面の左上を（1, 1）とするキャラクタ座標により指定します（文法編　ディスプレイ画面上の座標系参照）．

〈カーソルスイッチ〉はカーソルの表示状態を決めるスイッチで，1を指定するとカーソルは表示され，0を指定するとカーソルは表示されなくなります．ただし，これはプログラム実行中にのみ有効で，ダイレクトモードに戻ればカーソルは必ず表示されます．〈カーソルスイッチ〉が省略されたときは，1が仮定されます．

〈カーソルスタートライン〉,〈カーソルエンドライン〉は，カーソルの形を変えるためのパラメータでこの2つのパラメータによりカーソルの縦の幅を自由に変えることができます．カーソルのラインとは，カーソルの横方向のピクセルの並びのことで，〈カーソルスタートライン〉はカーソルの上端，〈カーソルエンドライン〉はカーソルの下端を示します．これらのパラメータは，0～254の範囲の整数で指定します．

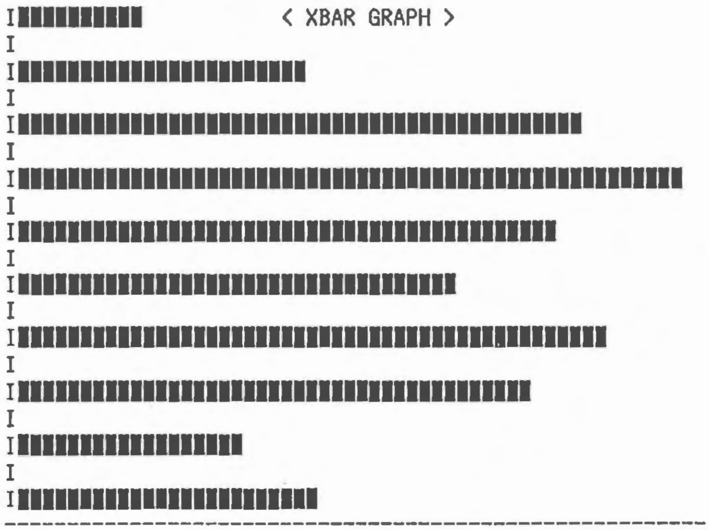


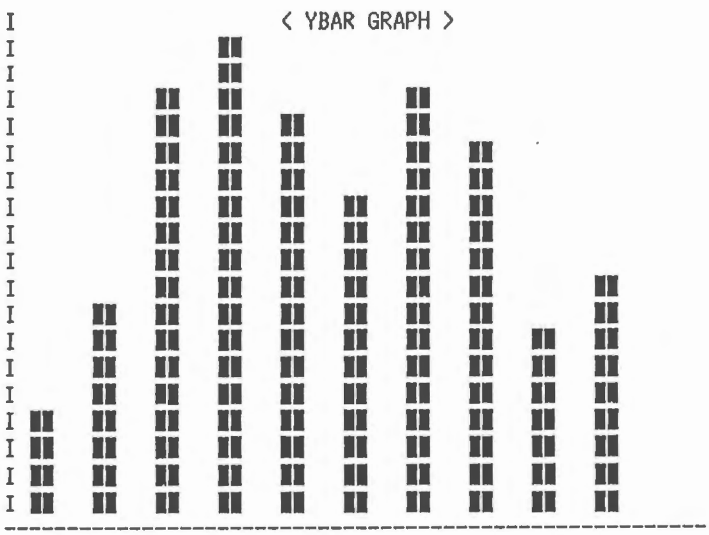
参照 SCREEN（関数）

サンプルプログラム

```
100 '-- character pattern --
110 SCREEN ,,1:WIDTH 80,25:CLS
120 DATA 10,23,45,53,43,35,47,41,18,24
130 FOR I=1 TO 10:READ DAT(I):NEXT I
140 GOSUB *GRAPH:GOSUB *XBAR
150 A$=INPUT$(1):CLS
160 GOSUB *GRAPH:GOSUB *YBAR:END
170 *GRAPH:'-- graph draw --
180  X=3:FOR Y=1 TO 19
190   LOCATE Y,X:PRINT "I":NEXT Y
200  LOCATE 20,3:PRINT STRING$(56,"-"):RETURN
210 *XBAR:LOCATE 1,25:PRINT "< XBAR GRAPH >"
220  FOR I=1 TO 10
230   LOCATE I*2-1,4:PRINT STRING$(DAT(I),CHR$(&H87))
240  NEXT I:RETURN
```

```
250 *YBAR:LOCATE 1,25:PRINT "< YBAR GRAPH >"
260  FOR I=1 TO 10
270   FOR J=19 TO 19-DAT(I)/3 STEP -1
280    LOCATE J,I*5:PRINT CHR$(&H87)
290    LOCATE J,I*5+1:PRINT CHR$(&H87)
300   NEXT J
310  NEXT I
320 LOCATE 23:RETURN
```





Ok

LOF
関数

機 能 ファイルの大きさを返す.

書 式 LOF (<ファイル番号>)

文 例 MAXREC=LOF (2)

解 説 <ファイル番号>により指定されたファイルがディスクファイルである場合には，LOF 関数はそのファイルの大きさをバイト数で返します．この値は，ファイルがランダムファイルであれば，その時点までにアクセスした最大のレコード番号にレコードサイズを乗じたものとなります．

指定されたファイルがRS-232C コミュニケーションファイルである場合には，LOF 関数は，通信バッファの残りのバイト数を返します．通信バッファは，BASIC の起動時に／C：パラメータによって設定された値となり，省略時は約256バイトとなります．

読取り動作（INPUT #，INPUT$など）が行われずに，割込みによるRS-232C 回線からの受信が続くと，やがて通信バッファが一杯になりエラーとなってしまいますので，このLOF 関数を使用して通信バッファを監視するとよいでしょう．

参 照 LOC

サンプルプログラム

```
100 OPEN "sdata1" FOR OUTPUT AS #1
110 PRINT #1,123!,234:CLOSE
120 OPEN "sdata1" FOR INPUT AS #1
130 OPEN "rdata1" AS #2 LEN=256
140 FIELD #2,128 AS A$,128 AS B$
150 PRINT "sdata1 ﾉ ﾌｧｲﾙｻｲｽﾞﾊ";LOF(1);"ﾊﾞｲﾄﾃﾞｽ"
160 PRINT "rdata1 ﾉ ﾌｧｲﾙｻｲｽﾞﾊ";LOF(2);"ﾊﾞｲﾄﾃﾞｽ"
170 CLOSE
```

```
run
sdata1 ﾉ ﾌｧｲﾙｻｲｽﾞﾊ 22 ﾊﾞｲﾄﾃﾞｽ
rdata1 ﾉ ﾌｧｲﾙｻｲｽﾞﾊ 0 ﾊﾞｲﾄﾃﾞｽ
Ok
```

LOG
関数

機　能　自然対数を返す.

書　式　LOG (<数式>)

文　例　PRINT LOG (35／9)

解　説　<数式> によって与えられた値の自然対数 (e を底とした対数) を返します.

また <数式> にはどんな型の数値を指定しても, LOG 関数は倍精度の値を返します.

参　照　EXP

サンプルプログラム

```
100 ' ｼﾞｮｳﾖｳ ﾀｲｽｳ ｦ ﾓﾄﾒﾙ
110  A=LOG(10)
120  F$="log10(##.#####^^^^)=##.#####^^^^"
130  INPUT L
140  A=LOG(10)
150  B=LOG(L)
160  PRINT USING F$;L,B/A
170 END
```

```
run
? 100
log10( 1.00000E+02)= 2.00000E+00
Ok
run
? 3541
log10( 3.54100E+03)= 3.54913E+00
Ok
run
? .6781
log10( 6.78000E-01)=-1.68770E-01
Ok
```

LPOS
関数

機能　現在のプリンタのヘッド位置を返す．

書式　LPOS（<ダミー引数>）

文例　LPOS (0)

解説　<ダミー引数> には，数値であれば何を指定してもかまいません．

プリンタバッファ内のプリンタヘッド位置を返します．LPOS 関数の返す値が，プリンタの物理的なヘッド位置であるとは限りません．

参照　WIDTH LPRINT

サンプルプログラム

```
100 WIDTH "lpt1:",255
110 FOR I=0 TO 20
120  IF LPOS(0)>=40 THEN LPRINT
130  LPRINT SQR(I),
140 NEXT I
150 END
 0              1              1.4142135623731
 1.7320508075689               2
 2.2360679774998               2.4494897427832
 2.6457513110646               2.8284271247462
 3              3.1622776601684
 3.3166247903554               3.4641016151378
 3.605551275464                3.741657386774
 3.8729833462074               4
 4.1231056256177               4.2426406871193
 4.3588989435407               4.4721359549996
```

LPRINT/LPRINT USING
ステートメント

機　能　プリンタにデータを出力する.

書　式　LPRINT〔〈式〉 ; / , 〔〈式〉 ; / , ・・・〕〕

LPRINT USING〈書式制御文字列〉;〈式〉〔 ; / , 〔〈式〉 ; / , ・・・〕〕

文　例　LPRINT A$, B

LPRINT USING "@####. ##" ; A$; B

解　説　〈式〉で表される数値変数，文字変数，数値定数，文字定数などのデータをプリンタに出力します．LPRINT 文，LPRINT USING 文は，出力先がプリンタであることを除けば，その機能は PRINT 文，PRINT USING 文とまったく同じですので，そちらを参照してください．

参　照　PRINT, PRINT USING

サンプルプログラム

```
100 A=123.456
110 LPRINT "NEC PC-100
120 LPRINT A
130 LPRINT USING "!";"NEC PC-100"
140 LPRINT USING "&      &";"NEC PC-100"
150 LPRINT USING "#####";A
160 LPRINT USING "#####.#";A
170 LPRINT USING "+####.#";-A
180 LPRINT USING "#####.#-";-A
190 LPRINT USING "**###.#";A
200 LPRINT USING "¥¥###.#";A
210 LPRINT USING "**¥##.#";A
220 LPRINT USING "#####,.#";A*10
230 LPRINT USING "#####,.#^^^^";A*10
240 LPRINT USING "¥¥###._-";A
250 LPRINT USING "#####";123456!
260 LPRINT USING "THIS IS A @.";"PEN"
270 END
```

```
NEC PC-100
 123.456
N
NEC PC-1
  123
  123.5
 -123.5
  123.5-
**123.5
 ¥123.5
*¥123.5
 1,234.6
 12345.6E-01
 ¥123.-
%123456
THIS IS A PEN.
```

LSET/RSET

ステートメント

機能　ランダムファイルバッファへデータを書き込む.

書式

LSET 〈文字変数〉= 〈文字列〉
RSET 〈文字変数〉= 〈文字列〉

文例

LSET A$= "PERSONAL"
RSET A$= "COMPUTER"

解説　Left SET, Right SET の意味で, ランダムファイルのレコードの FIELD 文で定義した文字変数に該当する領域にデータを格納します. 格納するデータは文字型データでなければならず, 数値データは MKI$, MKS$, MKD$のいずれかにより文字型データに変換しなければなりません.

〈文字列〉の文字数が FIELD 文で割り当てられた長さより短い場合, LSET では左詰め, RSET では右詰めでフィールドを満たします. このとき満たされなかった余りの部分にはスペースが入れられます. また, 〈文字列〉がフィールドより長い場合は, LSET, RSET とも右側の部分が失われます.

〈文字変数〉は, FIELD 文であらかじめ定義されていなければなりません. 定義されていない変数を用いるとエラーが発生します.

なお, LSET, RSET 両文は, 一定の長さを有する 〈文字変数〉 の中に, 〈文字列〉 を左詰め, 右詰めに代入するのに使うこともできます.

例えば,

A$= "ABC"
B$= "XXXXXXX"

のときに,

LSET B$=A$

を実行すると,

B$= "ABC␣␣␣␣"

となります (␣はスペースを意味します). RSET を使えば右詰めとなります. このとき, 〈文字変数〉 (上の例では B$) は単なるフィールドとして取り扱われるため, その内容は LSET, RSET 文を実行したときにすべてスペースにおきかえられます. 〈文字列〉 の方が 〈文字変数〉 より長いときは, 右側の部分が失われます.

参照　FIELD, PUT, MKI$／MKS$／MKD$

LSET/RSET

サンプルプログラム

```
100 OPEN "data" AS #1
110 FIELD 1,20 AS A$,10 AS B$
120 LINE INPUT "ﾋﾝﾒｲ ?";NAM$
130 INPUT "ﾀﾝｶ ";PRICE%
140 LSET A$=NAM$
150 RSET B$=MKI$(PRICE%)
160 PUT #1.1
170 CLOSE
```

```
run
ﾋﾝﾒｲ ?ﾌﾛｯﾋﾟｰ ﾃﾞｨｽｸ
ﾀﾝｶ ? 1500
Ok
```

MERGE
コマンド

機能　アスキー形式でセーブされたプログラムをメモリ上のプログラムにマージ（混合）する．

書式　MERGE <ファイルスペック>

文例　MERGE "B : ¥BAS¥TEST"

解説　<ファイルスペック>により指定されたプログラムファイルとメモリ上のプログラムを１つにして，メモリ上に置きます．ここで，指定されたファイルはアスキー形式でセーブされていなければなりません．そうでない場合には，エラーになります．

ファイル中のプログラムと，メモリ中のプログラムに同一の行番号があった場合には，メモリ中の行がファイル中の行に置き換わります．

MERGE コマンドは実行を終了すると，常にコマンドレベルに戻ります．マージを行った後，直ちにその混合されたプログラムを実行させたいときには，MERGE オプション付きの CHAIN 文を使用してください．

なお，<ファイルスペック>中のドライブ名，パス名が省略されたときには，それぞれカレントドライブ，カレントディレクトリが指定されます．

参照　SAVE, CHAIN

サンプルプログラム

```
list
100 PRINT "１００行"
110 PRINT "１１０行"
120 *ENTRY
130 PRINT "１３０行"
Ok
merge "test13"
Ok
list
100 PRINT "１００行"
110 PRINT "１１０行"
120 *ENTRY
130 PRINT "１３０行"
140 PRINT "１４０行"
150 PRINT "１５０行"
160 PRINT "１６０行"
Ok
```

MID$

ステートメント

機能　文字列の一部を置き換える.

書式　MID$ (〈文字列1〉, 〈式1〉〔, 〈式2〉〕) = 〈文字列2〉

文例　MID$ (A$, 3) = "ABC"

解説　〈文字列1〉の〈式1〉番目のバイト位置から〈式2〉バイトの文字を, 〈文字列2〉の最初から〈式2〉バイトの文字列で置き換えます. 〈式1〉は1から255の範囲, 〈式2〉は0から255の範囲の整数です.

〈式2〉を省略したり, 〈式2〉を〈文字列2〉のバイト数より多く指定したりした場合は, 〈文字列2〉のすべての文字が置換えに使用されます.

〈式1〉+〈式2〉−1の値が〈文字列1〉のバイト数より大きいときは, (〈文字列1〉のバイト数) − 〈式1〉+1の値を〈式2〉とします. したがって置き換えられる〈文字列2〉のバイト数も, (〈文字列1〉の文字数) − 〈式1〉+1バイトとなり, 置換え後の文字列の長さは〈文字列1〉の長さと同じになります.

また, 〈式1〉の値は〈文字列1〉のバイト数より大きくても, 0以下であってもいけません. したがって〈文字列1〉にヌルストリングを指定することはできません.

なお, 〈文字列1〉, 〈文字列2〉には2バイトキャラクタ (漢字など) も含むことができますが, 〈式1〉, 〈式2〉の値が2バイトキャラクタを2分しないよう, 語ごとの境界にじゅうぶん注意してください.

参照　MID$ (関数)

サンプルプログラム

```
100 DATA My name is !!!!!!!!
110 DATA Sato,Tanaka,Aizawa,Matubara
120 READ A$
130 FOR I=1 TO 4
140  READ NAM$:MID$(A$,12)=NAM$
150  PRINT A$
160 NEXT I

run
My name is Sato!!!!
My name is Tanaka!!
My name is Aizawa!!
My name is Matubara
Ok
```

MID$
関数

機　能　文字列の中から任意の長さの文字列を返す.

書　式　MID$ (〈文字列〉, 〈式 1 〉〔, 〈式 2 〉〕)

文　例　PRINT MID$ (A$, 2, 3)

解　説　〈文字列〉の〈式 1 〉番目のバイト位置から〈式 2 〉バイトの文字列を返します.

〈式 1 〉は 1 から255の範囲, また,〈式 2 〉は 0 から255の範囲の整数です.

〈式 2 〉を省略した場合や,〈式 1 〉+〈式 2 〉− 1 の値が〈文字列〉のバイト数より大きい場合は,〈文字列〉の〈式 1 〉番目のバイト位置より右すべての文字列を結果とします.

なお,〈式 1 〉が〈文字列〉のバイト数より大きければ, MID$関数は, ヌルを返します.

なお,〈文字列 1 〉,〈文字列 2 〉には 2 バイトキャラクタ（漢字など）も含むことができますが,〈式 1 〉,〈式 2 〉の値が 2 バイトキャラクタを 2 分しないよう, 語ごとの境界にじゅうぶん注意してください.

参　照　RIGHT$, LEFT$, MID$（ステートメント）

サンプルプログラム

```
100 '-- break of strings --
110  INPUT A$
120  PRINT A$;" = ";
130  FOR I=1 TO LEN(A$) STEP 2
140   PRINT MID$(A$,I,2);" + ";
150  NEXT
160  PRINT CHR$(8);CHR$(8);" "
170 END
```

```
run
? abcdefg
abcdefg = ab + cd + ef + g
Ok
```

MKDIR

コマンド

機　能　新しいディレクトリを作成する．

MKDIR〔〈ドライブ名〉〕〈パス名〉

文　例　MKDIR "B：¥DATA¥TEST.ASC"

解　説　〈ドライブ名〉で指定されるディスクに，〈パス名〉で指定される新しいディレクトリを作成します．〈ドライブ名〉が省略された場合には，カレントドライブが指定されます．

このコマンドにより，階層的ディレクトリ構造を構成することができます．

例えば，現在，カレントディレクトリがルートディレクトリだとします．

MKDIR "¥BIN"

はルートディレクトリの下に "¥BIN" という名のディレクトリを作成します．

MKDIR "¥BIN¥BAS"

は，今作成した "¥BIN" ディレクトリの更に下に，"¥BAS" という名のディレクトリを作成することができます．

現在のディレクトリ（カレントディレクトリ）がどこであろうと，〈パス名〉をルートから正しく指定すればどのディレクトリの下にでも新しいディレクトリを作成することができます．

詳しくは，BASIC ユーザーズガイドをご覧ください．

参　照　CHDIR，RMDIR

サンプルプログラム

```
100 PRINT "ディスクに書きこみます。適当なディスクを入れてください。":GOSUB TOMARE
110 PRINT "FILES":FILES
120 PRINT "ディレクトリを作ります。"
130 MKDIR "test"
140 CHDIR "test"
150 PRINT "files":FILES
160 PRINT "ディレクトリを作ります。"
170 MKDIR "test2"
180 PRINT "ディレクトリを作ります。"
190 MKDIR "test3"
200 PRINT "files":FILES
210 CHDIR ".."
220 END
230 TOMARE
240 PRINT "用意はいいですか。"
250 IF INKEY$="" THEN 250
260 RETURN
```

```
run
ディスクに書きこみます。適当なディスクを入れてください。
用意はいいですか。
FILES
A:¥
TEST1   .BAS     TEST2              TEST3   .BAS     TEST4   .BAS
TEST5   .BAS     TEST7              STEST   .DAT     TEST8
TEST10           TEST11
 352256 Bytes free
ディレクトリを作ります。
files
A:¥TEST
        .   <DIR>          ..  <DIR>
 351232 Bytes free
ディレクトリを作ります。
ディレクトリを作ります。
files
A:¥TEST
        .   <DIR>          ..  <DIR> TEST2        <DIR> TEST3        <DIR>
 349184 Bytes free
Ok
```

MKI$/MKS$/MKD$

関数

機能　各数値を内部表現に対応した文字コードに変換する.

書式　MKI$（<整数値>）

MKS$（<単精度実数値>）

MKD$（<倍精度実数値>）

文例　A$＝MKI$（128）

LSET B$＝MKS$（1.23）

RSET C$＝MKD$（3.1415926535898＃）

解説　これらの関数は，数値をランダムファイルバッファに対して LSET／RSET 文で書き込む際に使用します．数値から文字への変換は数値が持つ内部表現（２進数表現）の値をそのままそれに対応する文字コードにすることによって行われます（CVI／CVS／CVD 関数の項参照）.

整数値を２文字（２バイト）の文字列に変換します.

単精度実数値を４文字（４バイト）の文字列に変換します.

倍精度実数値を８文字（８バイト）の文字列に変換します.

なお，これとは逆に，ランダムファイルから読み込んだ文字型のデータを数値データに変換するには，CVI／CVS／CVD 関数を用います.

参照　CVI／CVS／CVD

サンプルプログラム

```
100 OPEN "RDATA" AS #1
110 FIELD #1,8 AS A$
120 A%=987:LSET A$=MKI$(A%):PUT #1
130 A!=132.456:LSET A$=MKS$(A!):PUT #1
140 A=123456789123#:LSET A$=MKD$(A):PUT #1
150 END
160 'このプログラムで書込むデータはＣＶＩ／ＣＶＳ／ ＣＶＤの
サンプルプログラムで読みだせます。"
```

MOUSE
ステートメント

機能　マウスの諸機能を設定する.

書式　MOUSE 0

MOUSE 1〔, Sx〕〔, Sy〕〔, 〈カーソルスイッチ〉〕

MOUSE 2, 〈AND マスク文字列〉, 〈XOR マスク文字列〉, 〈X 指示点〉, 〈Y 指示点〉

MOUSE 3, 〈方向〉, 〈移動比率〉

MOUSE 4, Sx1, Sy1, Sx2, Sy2

文例　MOUSE 0

MOUSE 1, 0, 0, 1

MOUSE 2, A$, X$, 7, 7

MOUSE 3, 0, 100

MOUSE 4, 100, 100, 400, 400

解説　マウスの諸機能を設定するステートメントです.

MOUSE 0 は, マウスカーソルの形状や移動比率, 移動範囲などを初期状態に戻すとともに, 表示をとりやめ, マウスの機能を OFF 状態にします. マウスの使用前, 使用後には, 必ずこの文を実行するようにしてください.

MOUSE 1 は, マウスカーソルの初期表示位置と, 表示／非表示を指定するステートメントです. スクリーン座標上の(Sx, Sy)で指定された座標位置にカーソルがセットされます. 〈カーソルスイッチ〉を 1 にするとカーソルが表示され, 0 にすると表示されません. ただし, 0 を指定したときはカーソルの表示がなされないだけで, マウスの機能そのものは ON 状態となっており, MOUSE 関数などでカーソル位置を得たりすることはできます. 〈カーソルスイッチ〉を省略すると 0 が指定されます. Sx, Sy を省略すると, マウスカーソルの現在位置が指定されたのと同じになります.

MOUSE 2 は, マウスカーソルの形状を任意に設定するためのステートメントです. マウスカーソルは, AND マスクと XOR マスクの 2 枚のパターンから成っており, その大きさは16×16ピクセルです. 各マウスは32バイトの文字列を連結することにより設定することができます. 実際の設定の仕方については BASIC ユーザーズガイドのマウスの項をご覧ください. 〈X 指示点〉, 〈Y 指示点〉は設定したマウスカーソルの指示点を決めるもので, それぞれ 0 ～15の値を指定します. MOUSE 1 ステートメントや MOUSE 関数などで指定する, あるいは返される座標は, この指示点に対して参照されます.

MOUSE 3 は, マウスカーソルの移動比率を指定するステートメントです. 〈移動比率〉には, 0 ～32767の範囲の整数を指定します. これは, マウスの移動距離に対するマウスカーソルの移動距離の割合を表すもので, 値の大きいほど, 画面上のマウスカーソルの動きは〝遅く〟なります. 〈方向〉に 0 を指定したときは X（水平）方向の移動比率を, 1 を指定したときは Y（垂直）方向の移動比

率をセットできます.

MOUSE 4 は，マウスカーソルの移動範囲を設定するステートメントです．スクリーン座標上に指定された 2 点 (Sx1，Sy1) と (Sx2，Sy2) を対角点とする四角形の内部がマウスカーソルの移動範囲となります.

注意：マウスを使用してグラフィックスを描いたりする場合は，LINE 文や CIRCLE 文などの描画コマンドを実行する直前に，必ず MOUSE 1,,, 0 を実行してマウスカーソルを消してください．これをしないと，描こうとするグラフィックスが一部欠けたりすることがあります．描画後は MOUSE 1,,, 1 を実行すればマウスカーソルは再び表示されます.

なお，このときブリンクカーソル（スクリーンエディタのカーソル）もプログラムのはじめなどで LOCATE ,, 0 を実行して消しておいてください.

サンプルプログラム

```
100 MOUSE 0:SCREEN 1,0,0,0
110 CLS
120 LOCATE 10:FOR I=1 TO 10:PRINT TAB(50);MID$("¦¦¦GOOL
¦¦¦",I,1):NEXT
130 A$=STRING$(32,CHR$(&HFF))
140 X$=CHR$(&H0,&H78,&H10,&H94,&HE8,&HB7,&HF8,&H9F,&H7C
,&H7F,&HF8,&H1F,&H80,&H1F,&HE0,&HF,&HC0,&H47,&H80,&H27,
&HC3,&H27,&HFC,&H4F,&HC0,&H4F,&HC0,&H4F,&H80,&H3F,&HE0,
&HF)
150 MOUSE 2,A$,X$
160 MOUSE 3,0,300:MOUSE 3,1,500
170 PRINT "MOUSE ｦ ｺﾞｰﾙ ﾏﾃﾞ ｲﾄﾞｳ ｼﾃ ｸﾀﾞｻｲ"
180 PRINT "ﾖｰｲ ";:FOR I=1 TO 5000:NEXT:BEEP :BEEP:PRINT
 "START"
190 MOUSE 1,0,270,1:TI=TIMER
200 IF MOUSE (0)>440 THEN *GOOL ELSE 200
210 *GOOL
220 T=TIMER-TI
230 PRINT USING "#### ﾋﾞｮｳ ｶｶﾘ ﾏｼﾀ";T
240 PRINT "MOUSE ﾉ ﾎﾞﾀﾝ ｦ ｵｽ ﾄ ﾂｷﾞﾉ DEMO ﾆ ｳﾂﾘﾏｽ｡"
250 IF MOUSE(2,0)=1 OR MOUSE(2,1)=1 THEN 260 ELSE 250
260 CLS
270 MOUSE 3.0,0:MOUSE 3,1,0
280 MOUSE 4,200,200,300,300
290 LINE (180,180)-(320,320),15,B,&H3333
300 PRINT "MOUSE ｦ ｳｺﾞｶｼﾃ ﾐﾃ ｸﾀﾞｻｲ"
310 PRINT "ｶｺﾞﾆ ﾊｲｯﾀ MOUSE ﾃﾞｽ｡"
320 PRINT "MOUSE ﾉ ﾎﾞﾀﾝ ｦ ｵｽ ﾄ DEMO ｦ ｵﾜﾘﾏｽ｡"
330 IF MOUSE(2,0)=1 OR MOUSE(2,1)=1 THEN 340 ELSE 330
340 MOUSE 0
```

MOUSE

関数

機 能　マウスの諸状態を返す.

書 式

```
MOUSE (0)
MOUSE (1)
MOUSE (2, 〈ボタン番号〉)
MOUSE (3, 〈ボタン番号〉)
MOUSE (4, 〈ボタン番号〉)
MOUSE (5, 〈ボタン番号〉)
MOUSE (6, 〈ボタン番号〉)
MOUSE (7, 〈ボタン番号〉)
MOUSE (8, 〈ボタン番号〉)
MOUSE (9)
MOUSE (10)
```

文 例

```
PRINT MOUSE (0)
A=MOUSE (6, 1)
```

解 説　マウスの諸状態を返します.

〈ボタン番号〉は，左のボタンを1，右のボタンを2で指定します．返される値が座標の場合は，すべてスクリーン座標系の座標となります.

MOUSE (0) は，マウスカーソルの現在のX（水平）座標を返します.

MOUSE (1) は，マウスカーソルの現在のY（垂直）座標を返します.

MOUSE (2，〈ボタン番号〉) は，指定された〈ボタン番号〉のボタンが押されているときに1を返し，離されているときに0を返します.

MOUSE(3，〈ボタン番号〉)は，最後にこの関数が呼び出されてから，次にこの関数が呼び出されるまでに〈ボタン関数〉のボタンが押された回数を返します.

MOUSE (4，〈ボタン番号〉) は，最後に〈ボタン番号〉のボタンが押されたときのマウスカーソルのX（水平）座標を返します.

MOUSE (5，〈ボタン番号〉) は，最後に〈ボタン番号〉のボタンが押されたときのマウスカーソルのY（垂直）座標を返します.

MOUSE(6，〈ボタン番号〉)は，最後にこの関数が呼び出されてから，次にこの関数が呼び出されるまでに〈ボタン番号〉のボタンが離された回数を返します.

MOUSE (7，〈ボタン番号〉) は，最後に〈ボタン番号〉のボタンが離されたときのマウスカーソル

のX（水平）座標を返します．

MOUSE（8，〈ボタン番号〉）は，最後に〈ボタン番号〉のボタンが離されたときのマウスカーソルのY（垂直）座標を返します．

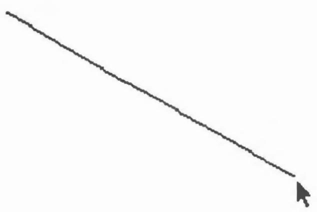
MOUSE（9）は，最後にこの関数が呼び出されてから，次にこの関数が呼び出されるまでにマウスカーソルが動いたX（水平）方向の移動距離を返します．

MOUSE（10）は，最後にこの関数が呼び出されてから，次にこの関数が呼び出されるまでにマウスカーソルが動いたY（垂直）方向の移動距離を返します．

サンプルプログラム

```
100 CLS
110 MOUSE 0
120 MOUSE 1,0,0,1
130 '
140 LOCATE 24,1:PRINT USING "ｹﾞﾝｻﾞｲ ﾉ ﾏｳｽ ﾉ ｲﾁ X=### Y=
 ###";MOUSE(0),MOUSE(1);
150 PRINT TAB(40);"ﾏｳｽ ﾉ ﾎﾞﾀﾝ ｦ ｵｾﾊﾞ ﾂｷﾞ ﾉ DEMO ﾆ ｲｷﾏｽ"
;
160 '
170 IF MOUSE(2,0)=4 OR MOUSE(2,1)=4 THEN 180 ELSE 140
180 CLS :PRINT "5 ﾋﾞｮｳ ｶﾝ ﾆ ﾅﾝｶｲ ﾎﾞﾀﾝ ｶﾞ ｵｾﾙ ｶ????":PRI
NT "ﾖｰｲ  ";:FOR I=1 TO 5000:NEXT:BEEP:BEEP:PRINT "START
"
190 A=MOUSE(3,0):B=MOUSE(3,1)
200 TI=TIMER
210 IF TI+5>TIMER THEN 210
220 BEEP:PRINT "ｵﾜﾘ":PRINT USING"ﾋﾀﾞﾘ ### ｶｲ  ﾐｷﾞ ### ｶ
ｲ";MOUSE(3,0),MOUSE(3,1);
230 PRINT TAB(40);"ﾏｳｽ ﾉ ﾎﾞﾀﾝ ｦ ｵｾﾊﾞ ﾂｷﾞ ﾉ DEMO ﾆ ｲｷﾏｽ"
;:FOR I=1 TO 1000:NEXT
240 IF MOUSE(2,0)=4 OR MOUSE(2,1)=4 THEN 260 ELSE 240
250 '
260 CLS
270 PRINT "line ｦ ﾋｷﾏｽ ｼﾃﾝ ﾃﾞ ﾋﾀﾞﾘ ｵｼ ｼｭｳﾃﾝﾃﾞ ﾐｷﾞ ｦ ｵｼﾃ
ｸﾀﾞｻｲ"
280 IF MOUSE(2,1)=4 THEN 290 ELSE 280
290 LINE(MOUSE(4,0),MOUSE(5,0))-(MOUSE(4,1),MOUSE(5,1))
300 PRINT TAB(40);"ﾏｳｽ ﾉ ﾎﾞﾀﾝ ｦ ｵｾﾊﾞ ｵﾜﾘ ﾏｽ";:FOR I=1 T
O 1000:NEXT
310 IF MOUSE(2,0)=4 OR MOUSE(2,1)=4 THEN 320 ELSE 310
320 MOUSE 0:END
```

```
line ｦ ﾋｷﾏｽ。ｼﾃﾝ ﾃﾞ ﾋﾀﾞﾘ ｦ ｵｼ ｼｭｳﾃﾝﾃﾞ ﾐｷﾞ ｦ ｵｼﾃｸﾀﾞｻｲ
                              ﾏｳｽ ﾉ ﾎﾞﾀﾝ ｦ ｵｾﾊﾞ ｵﾜﾘ ﾏｽ
```



MOUSE(n) ON/OFF/STOP

ステートメント

機　能　マウスによる割込みを許可，禁止，保留する．

書　式

MOUSE(〈事象番号〉) ON
MOUSE(〈事象番号〉) OFF
MOUSE(〈事象番号〉) STOP

文　例

MOUSE(1) ON

解　説

マウスを移動させたり，ボタンを押したりしたときに生ずる割込みを，許可（ON）するか，禁止（OFF）するか，保留（STOP）するかを設定します．

〈事象番号〉は，マウスに生じる割込みの要因が何であるのかを指定するもので，次の５つの事象についてセットできます．

１：マウスが移動した
２：ボタン１（左側）を押した
３：ボタン２（右側）を押した
４：ボタン１（左側）を離した
５：ボタン２（右側）を離した

MOUSE(n) ON は，〈事象番号〉で指定された事象についての割込みを許可します，このステートメント実行後は，指定した事象が起こるごとに割込みを発生し，ON MOUSE(n) GOSUB 文で定義されている処理ルーチンに分岐します．

MOUSE(n) OFF は，〈事象番号〉で指定された事象についての割込みを禁止します．このステートメント実行後は，事象が発生しても処理ルーチンへの分岐は起こりません．

MOUSE(n) STOP は，〈事象番号〉で指定された事象についての割込みを保留します．このステートメント実行後は，事象が発生してもそのことを覚えているだけで処理ルーチンへの分岐は起こりません．しかし，その後 MOUSE(n) ON によって割込みが許可されると，先程の事象の発生によって処理ルーチンに分岐します．

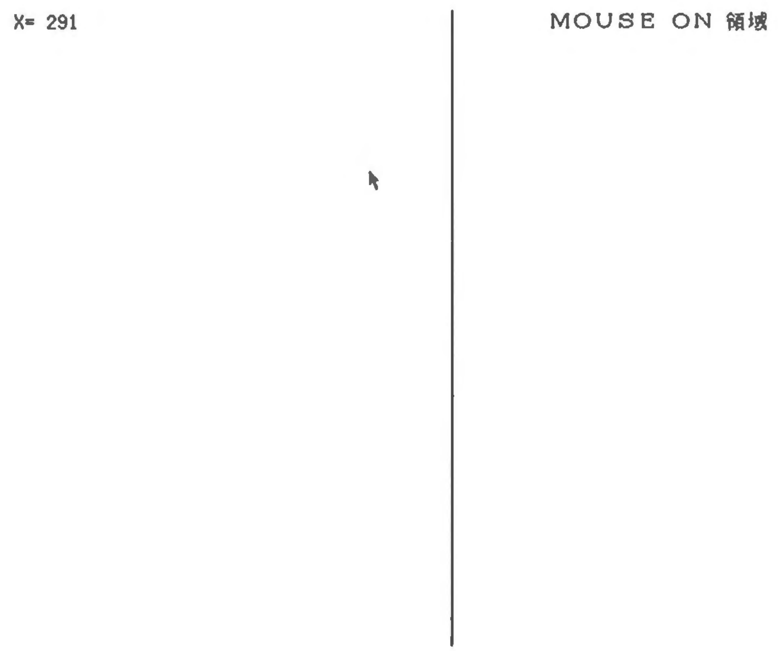
なお，割込み処理ルーチンに制御が移ると自動的に MOUSE(n) STOP（割込み保留）状態となりますので，割込み処理ルーチンの先頭で MOUSE(n) OFF 文あるいは MOUSE(n) STOP 文を実行する必要はありません．

サンプルプログラム

MOUSE(n)ON/OFF/STOP

サンプルプログラム

```
100 MOUSE 0:CLS:MOUSE 1,0,0,1:LINE(360,0)-(360,511)
110 LOCATE 1,50,0:PRINT"ＭＯＵＳＥ ＯＮ 領域"
120 MOUSE (1) ON:ON MOUSE(1) GOSUB 170
130 X=MOUSE(0):LOCATE 1,1,0:PRINT "X=";X
140 IF MOUSE(2,1)<>MOUSE(2,2) THEN MOUSE(1) OFF:CLS 3:END
150 IF X<360 THEN MOUSE(1) STOP ELSE MOUSE(1) ON
160 GOTO 130
170 BEEP:RETURN
```



NAME

コマンド

機能 ディスクファイルの名前を変える.

書式 NAME 〈ファイルスペック 1〉 AS 〈ファイルスペック 2〉

文例 NAME "B : SAMPLE. NEW" AS "B : SAMPLE. OLD"

解説 〈ファイルスペック 1〉で指定されるディスク上のファイルを〈ファイルスペック 2〉に変更します.

NAME コマンドはカレントディレクトリに対してのみ有効ですので,〈ファイルスペック 1〉と〈ファイルスペック 2〉中に,パス名を指定することはできません.カレントディレクトリとは別のディレクトリ中にあるファイルの名前を変更しようとする場合は,CHDIR コマンドを使って目的のディレクトリに移動してから NAME コマンドを実行するようにしてください.

また,〈ファイルスペック 1〉のファイル名に拡張子があるときには,それが".BAS"であってもこれを省略することはできません.

〈ファイルスペック 1〉および〈ファイルスペック 2〉中のドライブ名が省略されたときにはカレントドライブが仮定されます.

〈ファイルスペック 1〉および〈ファイルスペック 2〉中のドライブ名がそれぞれ異なるときには"Rename across disks"エラーが生じ,ファイル名の変更は行われません.

また,〈ファイルスペック 2〉と同じ名前のファイルが既に存在している場合は"File already exists"エラーが生じ,やはりファイル名の変更は行われません.

なお,ファイル名の変更を行うファイルはクローズされた状態でなければなりません.

参照 KILL, CHDIR

サンプルプログラム

```
100 OPEN "oldtest" FOR OUTPUT AS #1
110 FOR J=1 TO 10
120  WRITE #1.J
130 NEXT J:CLOSE
140 FILES:PRINT
150 NAME "oldtest" AS "newtest"
160 FILES
```

```
run
A:¥
TEST1   .BAS     TEST2              TEST3   .BAS     TEST4   .BAS
TEST5   .BAS     OLDTEST
 356352 Bytes free

A:¥
TEST1   .BAS     TEST2              TEST3   .BAS     TEST4   .BAS
TEST5   .BAS     NEWTEST
 356352 Bytes free
Ok
```

NEW
コマンド

機 能	メモリにあるプログラムを抹消し，すべての変数をクリアする.

書 式　NEW

文 例　NEW

解 説

NEW コマンドは，コマンドレベルにあるとき，新しいプログラムを入力する前に実行します.
NEW コマンドの実行が終わると，いつもコマンドレベルに戻ります.

NEW コマンドは，すべての変数(配列変数も含む)をクリアし，開かれているファイルがあるときにはそれもすべて閉じます.

DEF FN 文によるユーザー関数の定義もすべてクリアされます.
また，TRON コマンドによってトレースオンの状態になっているときにはこれをトレースオフにします.

参 照　ERASE

サンプルプログラム

```
list
100 ' NEW ｺﾏﾝﾄﾞ ﾉ ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ
110 '
120 ' NEW ｺﾏﾝﾄﾞ ﾊ ﾌﾟﾛｸﾞﾗﾑ ｦ
130 ' ｲﾆｼｬﾗｲｽﾞ ｼﾏｽ
140 '
150 PRINT " NEW ｺﾏﾝﾄﾞ ﾊ ﾌﾟﾛｸﾞﾗﾑ ｦ"
160 PRINT " ｲﾆｼｬﾗｲｽﾞ ｼﾏｽ."
170 END
Ok
run
 NEW ｺﾏﾝﾄﾞ ﾊ ﾌﾟﾛｸﾞﾗﾑ ｦ
 ｲﾆｼｬﾗｲｽﾞ ｼﾏｽ.
Ok
new
Ok
list
Ok
```

OCT$

関数

機能 10進数を8進数に変換し，その文字列を返す.

書式 OCT$（<数式>）

文例 PRINT OCT$（A）

解説 <数式> の値を8進数に変換して，その文字列を与える関数です.

<数式> の値の範囲は－32768から32767（または0から65535）までです.

これとは逆に，8進数を10進数として扱うには，8進数の前に〝&O″または〝&″をつけます．詳しくは，文法編　定数の項をご覧ください.

参照 VAL，HEX$

サンプルプログラム

```
100 '-- 10ｼﾝ --> 8ｼﾝ -->16ｼﾝ--
110 FOR I=0 TO 16
120   DE$=RIGHT$("   "+STR$(I),3)
130   OC$=RIGHT$("   "+OCT$(I),3)
140   HE$=RIGHT$("   "+HEX$(I),3)
150  PRINT "１６進：";DE$,"８進：";OC$,"１６進：";HE$
160 NEXT I

run
１０進：  0   8進：  0     １６進：  0
１０進：  1   8進：  1     １６進：  1
１０進：  2   8進：  2     １６進：  2
１０進：  3   8進：  3     １６進：  3
１０進：  4   8進：  4     １６進：  4
１０進：  5   8進：  5     １６進：  5
１０進：  6   8進：  6     １６進：  6
１０進：  7   8進：  7     １６進：  7
１０進：  8   8進： 10     １６進：  8
１０進：  9   8進： 11     １６進：  9
１０進： 10   8進： 12     １６進：  A
１０進： 11   8進： 13     １６進：  B
１０進： 12   8進： 14     １６進：  C
１０進： 13   8進： 15     １６進：  D
１０進： 14   8進： 16     １６進：  E
１０進： 15   8進： 17     １６進：  F
１０進： 16   8進： 20     １６進： 10
Ok
```

ON COM(n) GOSUB

ステートメント

機能 通信回線から割込みが発生したとき，分岐する処理ルーチンの開始行を定義する．

書式 ON COM(〈ポート番号〉) GOSUB 〈行番号〉

文例 ON COM(1) GOSUB 1000

解説 〈ポート番号〉で指定されたコミュニケーションポート（RS−232C 回線）に入力割込みがあったときに分岐する処理ルーチンの開始行を定義します．

〈ポート番号〉にはコミュニケーションポート番号を指定します．PC-100 は 1 チャネルのみですから 1 を指定してください．

〈行番号〉は分岐させる処理ルーチンの開始行番号です．〈行番号〉に 0 を指定したときは，このステートメントによる開始行番号定義は無効となります．

ON COM(n) GOSUB 文によって割込み処理ルーチンの開始行が定義されると，以後 COM(n) ON／OFF／STOP 各文によって割込みの許可，保留を設定することにより，割込み処理のコントロールをすることができます．詳しくは，COM(n) ON／OFF／STOP 文の項をご覧ください．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで RETURN 文によって行います．単に RETURN としたときは，中断した所から実行が再開され，RETURN の後ろに行番号を指定したときは，その行から再開されます．

コミュニケーションポートに割込みが発生して処理ルーチンにジャンプすると，自動的に COM(n) STOP 状態となり，処理ルーチン内での割込みは保留されます．そして RETURN 文が実行されると，同時に COM(n) ON 状態に戻ります．

参照 COM(n) ON／OFF／STOP，OPEN

サンプルプログラム

```
100 CLS
110 Y=13:X=1
120 OPEN "COM1:5E81NA" AS #1
130 ON COM(1) GOSUB RECIEVE
140 LOCATE 12,1:PRINT "受信"
150 LOCATE 1,1:PRINT "送信"
160 COM(1) STOP
170 A$=INKEY$:PRINT #1,A$;:PRINT A$;
180 COM(1) ON:GOTO 160
```

```
190 RECIEVE
200 X1=POS(0):Y1=CSRLIN
210 LOCATE Y,X
220 IF LOC(1)<>0 THEN PRINT INPUT$(LOC(1),#1);
230 X=POS(0):Y=CSRLIN
240 LOCATE Y1,X1
250 RETURN
```

送信
NEC PC-100

受信
PERSONAL COMPUTER

ON ERROR GOTO
ステートメント

機能 エラー割込みを可能にし，エラー処理ルーチンの開始行を定義する．

書式 ON ERROR GOTO <行番号>

文例 ON ERROR GOTO 1000

解説 エラー割込みが可能になると，エラーが検出されたとき指定されたエラー処理ルーチンへプログラムの制御が移ります．

エラー割込み機能を禁止するには，ON ERROR GOTO 0 を実行してください．その後のエラーに対してはエラーメッセージが表示され，プログラムの実行は停止します．

また，エラー処理ルーチン内に ON ERROR GOTO 0 がある場合には，エラー割込みの原因となったエラーメッセージが表示され，プログラムの実行は停止します．エラー処理ルーチン内では，エラー処理を行わないエラーに対しては，ON ERROR GOTO 0 を実行することをお勧めします．また，プログラムの最後には ON ERROR GOTO 0 を実行するようにしてください．

エラー処理ルーチンの実行中にエラーが起きた場合には，それに対するエラーメッセージが表示され，実行は停止します．エラー処理ルーチンの中でエラー割込みが起こることはありません．

なお，エラー処理ルーチンから抜け出すためには RESUME 文を使用します．エラーを起こした行，その次の行，任意の行のいずれかに制御を戻すことができます．

ON ERROR GOTO 文の効果は，別の ON ERROR GOTO 文を実行したり，CLEAR, RUN を実行したりすると消滅します．もちろん NEW, DELETE などによって処理ルーチンそのものがクリアされれば同様にその効果は失われます．エラー割込み処理は，他の割込み処理に対して最優先されます．

参照 RESUME

サンプルプログラム

```
100 ' X ﾉ y ｼﾞｮｳ ｦ ﾓﾄﾒﾙ "
110 ON ERROR GOTO *ERRORMES
120 *START:INPUT "X :";X:INPUT "y :";Y
130   IF X=0 THEN ERROR 250
140  Z=X^Y: PRINT "  y":PRINT " X =";Z
150 GOTO *START
160  *ERRORMES
170   IF ERR=250 THEN PRINT " ﾃｲｷﾞ ｻﾚﾏｾﾝ｡"
180   IF ERR=6 THEN PRINT "ｵｰﾊﾞｰ ﾌﾛｰ ﾃﾞｽ｡"
190 RESUME *START
```

```
run
X :? 2
y :? 8
  y
 X = 256
X :? 321
y :? 654
ｵｰﾊﾞｰ ﾌﾛｰ ﾃﾞｽ｡
X :? 0
y :? -2
 ﾃｲｷﾞ ｻﾚﾏｾﾝ｡
X :?
```

ON KEY(n) GOSUB

ステートメント

機能	キーによる割込みルーチンの開始行を定義する
書式	ON KEY(〈キー番号〉) GOSUB 〈行番号〉
文例	ON KEY(15) GOSUB 100

解説

〈キー番号〉で指定されたキーを押したときに起こる割込みによって分岐する処理ルーチンの開始行を定義します．

〈キー番号〉は，次のように指定します．

1～10：ファンクションキー（[f･1]～[f･10]）
11～14：カーソルキー（[↑]，[↓]，[←]，[→]）
15～20：ユーザー定義キー

ユーザー定義キーによる割込み設定を行うときは，あらかじめ KEY 文で15～20のキー番号のうち任意の番号に任意の文字列をセットしておいてください．

〈行番号〉は分岐させる処理ルーチンの開始行番号です．

〈行番号〉に 0 を指定したときは，このステートメントによる開始行番号定義は無効となります．ON KEY(n) GOSUB 文によって割込み処理ルーチンの開始行が定義されると，以後 KEY(n) ON／OFF／STOP 各文によって割込みの許可，禁止，保留を設定することにより，割込み処理のコントロールをすることができます．詳しくは KEY(n) ON／OFF／STOP 文の項をご覧ください．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで RETURN 文によって行います．単に RETURN としたときは，中断した所から実行が再開され，RETURN の後ろに行番号を指定したときは，その行から再開されます．

キーによる割込みが発生して処理ルーチンにジャンプすると，自動的に KEY(n) STOP 状態となり，処理ルーチン内での割込みは保留されます．そして RETURN 文が実行されると同時に KEY(n) ON 状態に戻ります．

ただし，処理ルーチン内で KEY(n) OFF 文が実行されたときには，KEY(n) ON 状態には戻りません．

参照 KEY(n) ON／OFF／STOP

ON KEY(n) GOSUB

サンプルプログラム

```
100 CLS
110 KEY 15," "
120 ON KEY(15) GOSUB HIZUKE
130 KEY(15) STOP
140 LOCATE 11,20,0:PRINT "只今の時刻は";TIME$;"です。"
150 LOCATE 13,10,0:PRINT "ｓｐａｃｅバーを押すと日付を
表示します。"
160 KEY(15) ON
170 GOTO 130
180 HIZUKE:KEY(15) OFF
190 LOCATE 11,20,0:PRINT "今日の日付は";DATE$;"です。"
200 LOCATE 13,10,0:PRINT "なにかｋｅｙを押すと元にもど
ります。      "
210 IF INKEY$="" THEN 210
220 CLS
230 KEY(15) ON:RETURN
```

今日の日付は08-30-1983です。

なにかｋｅｙを押すと元にもどります。

ON～GOSUB/ON～GOTO

ステートメント

機能　指定されたいずれかの行に分岐する．

書式　ON 〈式〉 GOSUB 〈行番号〉〔，〈行番号〉・・・〕

ON 〈式〉 GOTO 〈行番号〉〔，〈行番号〉・・・〕

文例　ON A GOSUB 100，200，300，400

ON A GOTO 100，200，300，400

解説　〈式〉の値がプログラムのどの行番号の行に分岐するかを決定します．〈行番号〉の並びは1から始まる数に対応します．例えば，〈式〉の値が3であったなら，行番号のリストの3番目の行が分岐先となるわけです．

ON～GOTO文では〈行番号〉の並びには各分岐先の行番号を指定します．

ON～GOSUB文では，〈行番号〉の並びには各サブルーチンの最初の行の行番号を指定します．サブルーチン内のRETURN文が実行されるとON～GOSUB文の次の行へプログラムの制御が移ります．

〈式〉にはなるべく整数型の変数を使用するようにしてください．実数型の変数を使用した場合はその値が四捨五入して用いられます．

〈式〉の値が負になった場合には，〝Illegal function call〞エラーが起こります，値が0または〈行番号〉の並びの個数より大きくなった場合には次の行へプログラムの制御が移ります．

サンプルプログラム

```
100 *ENTRY
110  INPUT "N=";N
120  SG=SGN(N)+1
130  ON SG GOTO *ZERO,*PLUS
140  PRINT "MINUS!":GOTO *ENTRY
150 *ZERO :PRINT "ZERO!":GOTO *ENTRY
160 *PLUS :PRINT "PLUS!":GOTO *ENTRY
```

```
run
N=? 5454
PLUS!
N=? -3564
MINUS!
N=? 0
ZERO!
N=?
Break in 110
Ok
```

ON MOUSE(n) GOSUB

ステートメント

機　能　マウスによる割込みルーチンの開始行を定義する.

書　式　ON MOUSE(<事象番号>)　GOSUB <行番号>

文　例　ON MOUSE(1) GOSUB 200

解　説　マウスを移動させたり，ボタンを押したりしたときに起こる割込みによって分岐する処理ルーチンの開始行を定義します.

<事象番号>は，マウスに生ずる割込みの要因が何であるかを指定するもので，次の5つの事象についてセットできます.

1：マウスが移動した
2：ボタン1（左側）を押した
3：ボタン2（右側）を押した
4：ボタン1（左側）を離した
5：ボタン2（右側）を離した

<行番号> は分岐させる処理ルーチンの開始行番号です.

<行番号>に0を指定したときは，このステートメントによる開始行番号定義は無効となります. ON MOUSE(n) GOSUB 文によって割込み処理ルーチンの開始行が定義されると，以後 MOUSE (n) ON／OFF／STOP 各文によって割込みの許可，禁止，保留を設定することにより，割込み処理のコントロールをすることができます. 詳しくは MOUSE(n) ON／OFF／STOP 文の項をご覧ください.

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで RETURN 文によって行います. 単に RETURN としたときは，中断した所から実行が再開され，RETURN の後ろに行番号を指定したときは，その行から再開されます.

マウスによる割込みが発生して処理ルーチンにジャンプすると，自動的に MOUSE(n) STOP 状態となり，処理ルーチン内での割込みは保留されます. そして RETURN 文が実行されると同時に MOUSE(n) ON 状態に戻ります.

ただし，処理ルーチン内で MOUSE(n) OFF 文が実行されたときには，MOUSE(n) ON 状態には戻りません.

参　照　MOUSE(n) ON／OFF／STOP

サンプルプログラム

```
100 MOUSE 0:CLS
110 FOR I=2 TO 5:MOUSE(I) ON:NEXT
120 ON MOUSE(2) GOSUB 170
130 ON MOUSE(3) GOSUB 180
140 ON MOUSE(4) GOSUB 190
150 ON MOUSE(5) GOSUB 200
160 PRINT A;:A=A+1:GOTO 160
170 N=2:GOSUB 210:SOUND 1000,40:RETURN
180 N=3:GOSUB 210:SOUND 2000,30:RETURN
190 N=4:GOSUB 210:SOUND 3000,20:RETURN
200 N=5:GOSUB 210:SOUND 4000,10:RETURN
210 COLOR 14:PRINT "マウス割込み（";N;"）が発生";:COLOR 15:R
ETURN
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 マウス割込み（ 2 ）が発生 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 マウス割込み（ 4 ）が発生 56 57 58 59 60 61 62 63 64 65 66 67
68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86
マウス割込み（ 3 ）が発生 87 88 89 90 91 92 93 94 95 96 97 98 99
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115
116 117 118 マウス割込み（ 5 ）が発生 119 120 121 122 123 124 125 126
127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142
143 144 145 146 147 148 149 150 151 152
```

ON TIMER(n) GOSUB

ステートメント

機能 タイマの割込み条件を設定し，割込み処理ルーチンの開始行を定義する．

書式 ON TIMER(<数式>) GOSUB <行番号>

文例 ON TIMER(10) GOSUB 100

解説 このステートメントは，割込みがかかる時間間隔を設定し，割込みがかかったときに分岐する処理ルーチンの開始行番号を定義します．

<時間> は，1秒単位で，1～65535までの数値で指定します．

<行番号> は，分岐させる処理ルーチンの開始行番号です．

<行番号>に 0 を指定したときは，このステートメントによる開始行番号定義は無効となります．ON TIMER(n) GOSUB 文によって割込み処理ルーチンの開始行が定義されると，以後，TIMER (n) ON／OFF／STOP 各文によって割込みの許可，保留を設定することにより，割込み処理のコントロールをすることができます．詳しくは TIMER (n) ON／OFF／STOP 文の項をご覧ください．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで RETURN 文によって行います．単に RETURN としたときは，中断した所から実行が再開され，RETURN の後ろに行番号を指定したときは，その行から再開されます．

タイマによる割込みが発生して処理ルーチンにジャンプすると，自動的に TIMER(n) STOP 状態となり，処理ルーチン内での割込みは保留されます．そして RETURN 文が実行されると同時に TIMER(n) ON 状態に戻ります．

参照 TIMER ON／OFF／STOP

サンプルプログラム

```
100 ON TIMER(20) GOSUB OVER
110 Z$="mnt150o3c2c4.c8c2e-4.d8d4.c8c4.o2b8o3c1"
120 PRINT TAB(10);"ＴＹＰＥの練習"
130 PRINT TAB(15);"大文字で入力してください"
140 PRINT :PRINT "ヨーイ":FOR I=1 TO 3000:NEXT:BEEP:PRINT "
スタート"
150 FOR I=1 TO 20
160 A$=A$+CHR$(RND*26+64)
170 NEXT I
180 PRINT A$
```

```
190 BEEP
200 TIMER ON
210 C$=INPUT$(1):PRINT C$;:B$=B$+C$
220 IF LEN(B$)>=26 THEN 230 ELSE 210
230 IF A$=B$ THEN *ATARI ELSE *HAZURE
240 END
250 *ATARI
260 BEEP:PRINT:PRINT "アタリー":END
270 *HAZURE
280 BEEP:BEEP:PRINT :PRINT "ハズレー"
290 PLAY Z$:END
300 *OVER
310 PRINT :PRINT "ｔｉｍｅ　ｏｖｅｒ":PLAY Z$
320 END
```

```
run
          ＴＹＰＥの練習
               大文字で入力してください

ヨーイ
スタート
IXUDIMKPNRCASODDDLCH
IXUDIMKPNRCAS
ｔｉｍｅ　ｏｖｅｒ
Ok
```

OPEN
ステートメント

機　能　ファイルを開く.

書　式　OPEN 〈ファイルスペック〉〔FOR 〈モード〉〕AS #〈ファイル番号〉

文　例　OPEN　"B : ¥DATA¥TEST" FOR INPUT AS # 1

解　説　〈ファイルスペック〉で指定されたファイルを，指定された〈ファイル番号〉でオープンします．以後，オープンされたファイルへの入出力は，〈ファイル番号〉を指示することにより行います．

〈ファイルスペック〉中のドライブ名，パス名が省略されたときには，それぞれカレントドライブ，カレントディレクトリが仮定されます．〈ファイルスペック〉には，ディスクファイル名のほか，各種のデバイス名も使用することができます．

なお，ファイルの階層構造により，異なるディレクトリ中に同名のファイルを格納しておくことができますが，これらのファイルを同時にオープンすることはできませんので注意してください．

〈モード〉は，ファイルのアクセス方法を指定します．モードには次の 4 種類があります．

INPUT	入力モード．既存のファイルから入力を行うことを指示します．
OUTPUT	出力モード．新しくファイルを作り，出力を行うことを指示します．
APPEND	追加モード．既存のファイルの終わりに追加出力を行うことを指示します．
省略した時	FOR 節が省略されると，ランダムアクセスを行うことを指示します．

入力モードおよび追加モードでは，指定されたファイルが存在しないと，"File not found" エラーとなります．

出力モードでは，常に指定された名前のファイルを新しく作り，同一名のファイルがあった場合には新たに作られます．

デバイス名を〈ファイルスペック〉に使用する場合，デバイスによって入力，出力のモードが決まっていますので，違うモードでオープンしないよう注意してください．デバイス名とそのモードについては "付録　デバイス名" をご覧ください．

〈ファイル番号〉は，1 から15までの値を用いることができますが，BASIC を起動するときに／F：パラメータで指定したファイルの数を越えてはなりません．また，既にオープンされているファイルが利用している番号を用いることはできません．

ここで，〈ファイル番号〉とは，入出力のためのバッファ領域に対してつけた固有の番号のことです．OPEN 文は，〈ファイルスペック〉によって指定されたファイルをこのバッファ領域の番号に結合する働きをします．そして，いったんこの結合関係が成立すると，以後，ファイルへの入出力はこのバッファ領域を通じて行われることになるのです．

これとは逆に，CLOSE 文は，このバッファ領域とファイルとの結合関係を解除する働きをもっています．

なお，RS-232C 回線をコミュニケーションファイルとしてオープンする際には，〈ファイルスペック〉にデバイス名 COMn：を指定すればよいのですが，通信形式などの指定が必要となります．これについては別項 OPEN "COMn：をご覧ください．

参照　CLOSE，VARPTR，FIELD，OPEN "COMn：

サンプルプログラム

```
100 OPEN "sesame" FOR OUTPUT AS #1
110 PRINT #1,"ﾋﾗｹ ｺﾞﾏ"
120 PRINT #1,"ﾄｼﾞﾛ ｺﾞﾏ
130 CLOSE
140 OPEN "sesame" FOR INPUT AS #1
150 FOR I=1 TO 2
160 INPUT#1,A$
170 PRINT A$
180 NEXT
190 CLOSE

run
ﾋﾗｹ ｺﾞﾏ
ﾄｼﾞﾛ ｺﾞﾏ
Ok
```

OPEN"COMn:〜"

ステートメント

機能　RS-232C コミュニケーションファイルを開く.

書式　OPEN "COM 〈ポート番号〉: 〈ボーレート〉〈パリティ〉〈ワード長〉〈ストップビット〉〈コントロール〉〈モード〉" AS # 〈ファイル番号〉

文例　OPEN "COM1 : 8N82XA" AS #1

解説　RS-232C 回線の通信形式を設定し，コミュニケーションファイルとしてオープンします.

〈ファイル番号〉に関しては通常の OPEN 文と同様ですから，OPEN 文の項を参照してください. なお，コミュニケーションファイルは形としてはシーケンシャルファイルですが，入出力のモードはランダムファイルのように入出力兼用となりますので，入出力のモードの設定はとくに必要はありません.

〈ポート番号〉には，コミュニケーションポート番号を指定します. PC-100 は 1 チャネルのみですから 1 を指定してください.

〈ボーレート〉〜〈モード〉は，通信形式の設定を行うためのパラメータで，それぞれ 1 文字の英字または数字で指定します. それぞれの意味と指定の仕方は次のとおりです.

〈ボーレート〉 ……………通信速度（bps）の設定

2：150bps	6：2400bps
3：300bps	7：4800bps
4：600bps	8：9600bps
5：1200bps	9：19200bps

〈パリティ〉 ………………パリティビットの設定

N：パリティなし
E：偶数パリティ
O：奇数パリティ

〈ワード長〉 ………………1 ワードあたりのビット数の設定

7：7 ビット長
8：8 ビット長

〈ストップビット〉 ………ストップビット数の設定

1：1 ビット
2：1.5ビット
3：2 ビット

〈コントロール〉 …………制御方式の設定

X：Xon／Xoff コントロールを行う
N：Xon／Xoff コントロールを行わない

〈モード〉 ……………………アスキーモード／バイナリモードの設定
S：7ビットアスキーモード（SI／SO 制御を行う）
A：8ビットアスキーモード（SI／SO 制御をしない）
B：バイナリモード

これらのパラメータは省略することもできますが，書式に示した並び順の途中のパラメータを省略することはできません．つまり，

OPEN "COM1：3"
OPEN "COM1：8N82"

のように途中のパラメータ以降のすべてを省略することは可能ですが，

OPEN "COM1：38XS"

のように途中が抜けているような指定をすることはできません．省略されたパラメータは，最後に設定された値をとります．なお，起動時の初期値は，

"8N81XA"

に設定されています．

サンプルプログラム

```
100 OPEN "COM1:5E81NA" AS #1
110 INPUT"質問は？",A$
120 IF A$="End" THEN END
130 PRINT #1,A$
140 INPUT #1,B$
150 PRINT "答えは？";B$
160 GOTO 110
run
質問は？What is your name?
答えは？My name is Nobu.
質問は？
```

OPTION BASE

ステートメント

機能 配列の添字の下限を宣言する.

書式 OPTION BASE |0|1|

文例 OPTION BASE 1

解説 配列の添字の下限を 0 または 1 にすることを宣言します.

ふつう，この宣言が用いられない場合には，下限は 0 ですが，OPTION BASE 1 を実行すると下限は 1 になり，以後，配列の添字として 0 が用いられると，"Subscript out of range" エラーが起こるようになります.

この宣言は，プログラム中で配列変数が宣言，または引用された後に行うことはできません．また，CHAIN 文によって配列変数が引き渡された側のプログラム中でこの宣言を行うこともできません．このような場合には "Duplicate Definition" エラーになります.

OPTION BASE は一度宣言すると，RUN あるいは CLEAR を行うまでは，解除，変更することができません.

サンプルプログラム

```
100 OPTION BASE 0
110 DIM A(1)
120 PRINT "A(0) =";A(0),"A(1) =";A(1)
130 '
140 CLEAR:OPTION BASE 1
150 '-- error occur --
160 PRINT "A(0) =";A(0),"A(1) =";A(1)
```

```
run
A(0) = 0        A(1) = 0
A(0) =
Subscript out of range in 160
Ok
```

OUT
ステートメント

機能 出力ポートに1バイトのデータを送る.

書式 OUT <ポート番号>, <式>

文例 OUT &H22, &H80

解説 <ポート番号> は出力ポートの番号，そして <式> は出力する1バイトのデータです．どちらのパラメータも&H0～&HFF（0～255）までの整数で指定します．

サンプルプログラム

```
100 PRINT "何か ｋｅｙ を押すと音が変わります。"
110 OUT &H22,&H80
120 IF INKEY$="" THEN 120
130 'ｵﾄｦ ｶｴﾙ
140 FOR I=1 TO 500
150 OUT &H22,&HC0:FOR J=1 TO I/15:NEXT:OUT &H22,&H40
160 NEXT

run
何か ｋｅｙ を押すと音が変わります。
Ok
```

PAINT

ステートメント

機能

指定された境界色で囲まれた領域を，指定された色で塗る，あるいは指定されたタイルパターンで埋める．

書式

1）PAINT ｜ (Wx, Wy) ｜ 〔,〈領域色〉〕〔,〈境界色〉〕
　　　　　｜ STEP (x, y) ｜

2）PAINT ｜ (Wx, Wy) ｜ 〔〈タイル文字列〉〕〔,〈境界色〉〕
　　　　　｜ STEP (x, y) ｜

文例

1）PAINT (10, 10), 12, 4

2）PAINT (255, 120), TILE$, 4

解説

〈境界色〉で囲まれた領域内全部を，ワールド座標点（Wx，Wy）を出発点として，

1）〈領域色〉が指定されているときは，ペイント（色を塗りつぶす）．

2）〈タイル文字列〉が指定されているときは，タイリング（模様で埋めつくす）．

を行います．ここで，STEP をつけると相対座標による指定となります．

1）ペイント

〈領域色〉，〈境界色〉はともにパレット番号で指定を行います．〈領域色〉が省略された場合には，COLOR 文で指定されたフォアグラウンドカラーが用いられます．〈境界色〉が省略された場合には，〈領域色〉の指定と同じパレット番号が〈境界色〉として用いられます．

(Wx，Wy) は塗り始める座標を指定します．もしこの点が，既に指定された境界色と同じ色であった場合には，PAINT は何の画面操作も行いません．PAINT はビューポート内でのみ働きますので，ビューポートの境界は PAINT 動作の境界とみなされます．また，(Wx，Wy) がウィンドウの外にある場合には，対応するスクリーン座標はビューポート外になるために，“Illegall function call” エラーとなります．

また〈境界色〉によって囲まれた領域が完全に閉じていないと，その箇所から領域色がもれ出して外の領域まで塗ってしまうことがありますので注意してください．

2）タイリング

〈タイル文字列〉は，タイリングに用いられる基本タイルの模様と大きさを決定する文字列です．タイルの大きさは，横方向は 8 ピクセル分と決められていますが，縦方向の長さは〈タイル文字列〉の長さで指定することができます．縦方向の長さが n ピクセルのタイルを指定するためには白黒モードで n バイト，カラーモードで 4 ×n バイトの長さを必要とします．ただし，〈タイル文字列〉の最大バイト数は，64です．

タイルの模様は，〈タイル文字列〉に対応するキャラクタコードの 2 進数表現（ビットパターン）により表現されます．〈タイル文字列〉の指定方法はカラーモードと白黒モードで異なります．

白黒モードでは，1バイトを横8ピクセルのラインに対応させて指定します．<タイル文字列>中のキャラクタコードは，8ピクセルのうちのどのピクセルをセットし，どのピクセルをリセットするかを決定します．つまり，2進数表現で1のビットに対応するピクセルはセット，0に対応するピクセルはリセットされることになり，これが横8ピクセルの白黒のドットパターンを表すことになります．

<タイル文字列> がnバイトの長さであれば，その文字列の表す領域は，このようにして決定される横8ピクセルのパターンをnライン分だけ縦に並べたものです．

例） CHR$ (&HAA)＋CHR$ (&H55)

16進数表現	2進数表現	ドットパターン
&HAA	10101010	●○●○●○●○
&H55	01010101	○●○●○●○●

上例は2バイトからなる簡単なタイル文字列の例です．この2バイトは互いに1であるビットが全く逆になっています．このパターンを基本タイルとして指定された領域を埋めれば，細かい市松模様になりますが，PC-100のドットが大変細かいために，その模様はほとんどわからず，灰色のように見えます．なお，上の例は，CHR$(&HAA，&H55) のようにも書けます(CHR$関数参照)．

カラーモードでは，4バイトを横8ピクセルのラインに対応させて指定します．<タイル文字列>中のキャラクタコードは，8ピクセルのうちのどのピクセルにどのパレット番号を対応させるかを決定します．つまり，キャラクタコード4バイトが8ピクセル分のカラーのドットパターンを表すことになります．
したがって，横8ピクセルのパターンを縦にnライン並べる場合，カラーモードでは <タイル文字列> が4×nバイト必要となります．

例） CHR$ (&HEA, &HE6, &H1E, &H01)

16進数表現	2進数表現	ドットパターン
&HEA	11101010	33345678
&HE6	11100110	
&H1E	00011110	
&H01	00000001	
	↓↓↓↓↓↓↓↓	
	33345678	

上例は4バイトから成るタイル文字列の例で，これにより横ピクセルだけの1ラインパターンを作ることができます．この4バイトを並べた状態で，各ビットを上の図のように〝たて〟に読むと，頭から順に，3，3，3，4……となっていますが，これが各ピクセルのパレット番号

となります．したがって，このパターンでタイリングを行うと，各ピクセルごとに異った色をもつ8ピクセルのラインで領域が埋められます．この状態でPALETTE文によってパレットへのカラーコードの割付けを変化させると，もちろん各ピクセル単位で色が変わることになります．

カラーモードでのタイル文字列の作り方についての詳細は，BASICユーザーズガイドで解説していますのでそちらをご覧ください．

注意：既にタイリングされている領域を異なるパターンでタイリングしようとすると時間がかかったり，スタックを消費するために〝Out of memory〞エラーになったりすることがあります．

また，〈タイル文字列〉中に，連続した3ライン分以上のヌルキャラクタ（白黒モードでは3キャラクタ，カラーモードでは12キャラクタ）があると，〝Illegal function call〞エラーとなります．例えば，白黒モードで，

CHR$（&HAA，&H5A，&H0，&H0，&H0）

はエラーとなります．なお，

CHR$（&H0，&HAA，&H5A，&H0，&H0）

あるいは単に，

CHR$（&H0）

のような場合も，タイルパターンが繰り返されることによって3ライン分のヌルキャラクタが上下で連続することになりますので，やはりエラーが生じます．ご注意ください．

参　照　COLOR，PALETTE，VIEW，CHR$

サンプルプログラム

```
100 DEFINT X,Y
110 CLS:LOCATE 1,1,0
120 A$=CHR$(&HFF,&H0,&H10,&HFF)
130 B$=CHR$(&HFF,&H0,&H38,&HFF)
140 C$=CHR$(&HFF,&H0,&HFE,&HFF)
150 D$=CHR$(&HFF,&H0,&H7C,&HFF)
160 D$=B$
170 F$=D$
180 G$=CHR$(&HFF,&H0,&HC6,&HFF)
190 H$=CHR$(&HFF,&H0,&H0,&HFF)
200 Z$=A$+B$+C$+D$+E$+F$+G$+H$
```

```
210 X=180:Y=256:GOSUB *STAR
220 PAINT(X,Y),14,9
230 X=540:Y=256:GOSUB *STAR
240 PAINT(X,Y),Z$,9
250 END
260 *STAR
270 FOR I=0 TO 360 STEP 72
280 PRESET (X,Y)
290 T1=I+165:T2=T1+30
300 T1=T1 MOD 360:T2=T2 MOD 360
310 DRAW"c9;ta=i;bu128;ta=t1;nu97;ta=t2;nu97"
320 NEXT
330 RETURN
```

PALETTE

ステートメント

機　能　パレットにカラーコードを割り付ける．

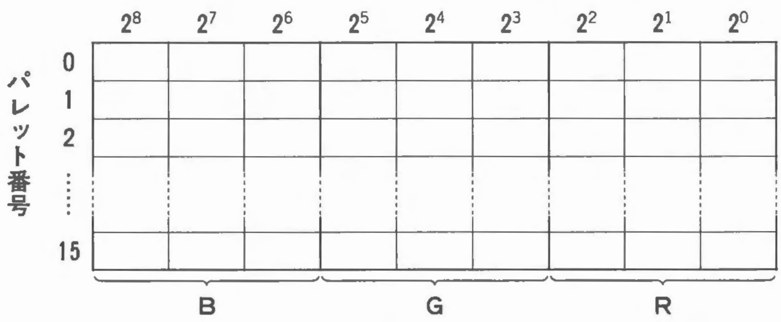
書　式　PALETTE〔〈パレット番号〉〕〔，〈カラーコード〉〕

文　例　PALETTE 6，&O246

解　説　〈パレット番号〉で指定されるパレットに，〈カラーコード〉で指定される色を設定します．

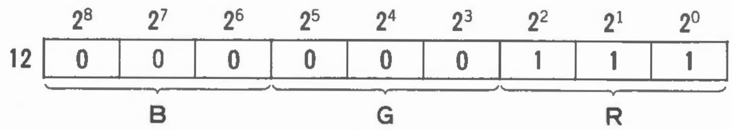
〈パレット番号〉は 0 ～15，〈カラーコード〉は&O0～&O777（0～511）の範囲の整数です．

パレットは全部で16個あり，1 個につき 9 ビットから成っています．



9 ビットは 3 ビットずつに区分され，下位の 3 ビットは赤 (R)，中位の 3 ビットは緑 (G)，上位の 3 ビットは青（B）に対応し，さらに各 3 ビットは 8 段階の輝度（明るさ）を表しています．

例えば，パレット番号12のパレットの初期状態を図示してみると，次のようになっています．



これは，B と G の色が輝度がゼロで，R の色だけ輝度が $(111)_2$ つまり $(7)_8$ で〝明るい赤〞を表すことになるわけです．ここで 8 進表現を用いたのは，3 ビットごとに区切って表すのに便利だからです．上のパレットの状態を PALETTE 文で書けば次のようになります．

```
PALETTE 12, &O007
```

これと同様にパレット番号15のパレットの初期状態は，

```
PALETTE 15, &O777
```

と表せます．つまり，R，G，B すべての色がいちばん高輝度となっていて，〝明るい白〞を表現しているわけです．

このようにR, G, B各8段階の輝度を表すことができますから，各パレットで表現できる色の数は8×8×8＝512色となります．詳しいパレットの設定方法についてはBASICユーザーズガイドをご覧ください．

なお，PALETTE文をパラメータなしで実行すると，各パレットの割付けは初期状態に戻ります．また，初期状態では，パレット番号15がフォアグラウンドカラー，パレット番号0がバックグラウンドカラーに，それぞれ対応しています．フォアグラウンドカラー，バックグラウンドカラーを変えるためにはCOLOR文を使用してください．

参照　COLOR

サンプルプログラム

```
100 CLS:LOCATE 20,20,0
110 PA=0
120 FOR Y=0 TO 384 STEP 128
130  FOR X=0 TO 540 STEP 180
140    LINE (X,Y)-STEP(180,128),PA,BF
150   PA=PA+1
160  NEXT
170 NEXT
180 FOR CO=0 TO 511
190    PALETTE (CO MOD 16),CO
200       FOR I=1 TO 200 :NEXT
210 NEXT
220 PALETTE
```

PALETTE USING

ステートメント

機　能　配列変数により，全パレットにカラーコードを割り付ける.

書　式　PALETTE USING 〈配列変数名〉〔(〈要素〉)〕

文　例　PALETTE USING E%

解　説　〈配列変数名〉で指定される配列変数にあらかじめ代入されている16種類のカラーコードを，いちどにすべてのパレットに割り付けるためのステートメントです.

〈配列変数名〉は，16以上の配列要素をもつ整数型の配列変数の名前です. この配列変数には，あらかじめ16種類以上のカラーコードを代入しておく必要があります.

〈要素〉は，上記の配列変数中の各要素に代入されたカラーコードを何番目の要素からパレットに割り付けるのかを指定するものです. つまり，〈要素〉番目から〈要素〉+15番目までのカラーコードが，パレット番号0からパレット番号15までのパレットに割り付けられるわけです.

例えば,

PALETTE USING F%(5)

とした場合，次の16の文をいちどに実行したのと等価となります.

PALETTE 0, F%(5)
PALETTE 1, F%(6)
PALETTE 2, F%(7)
⌇
PALETTE 15, F%(20)

〈要素〉が省略されたときには，そのときの配列の下限 (OPTION BASE 文参照) が用いられます.

なお，配列変数の各要素に代入するカラーコードとして－1を指定すると，その要素に対応するパレットに割り付けられるカラーコードは，以前割り付けたままに保たれます.

参　照　PALETTE, DIM, OPTION BASE

サンプルプログラム

```
100 SCREEN 1,0,0,0:LOCATE 1,1,0
110 DIM A%(511)
120 FOR I=0 TO 511
130  A%(I)=512*RND
140 NEXT
```

```
150 FOR Y=0 TO 480 STEP 32
160  LINE(0,Y)-STEP(719,31),C,BF
170 C=C+1:NEXT Y
180 FOR I=0 TO 496 STEP 16
190  PALETTE USING A%(I):C=0
200  A$=INKEY$:IF A$="" THEN 200
210 NEXT I
220 CLS 3:PALETTE
```

PEEK
関数

機 能 メモリ上の指定番地の内容を読み出す.

書 式 PEEK (<アドレス>)

文 例 A=PEEK (&HA000)

解 説 <アドレス>で指定されるオフセットアドレスのメモリの内容を返します. メモリ上の実際のアドレスは, 最後に実行されたDEF SEG文によって指定されたセグメントベースに, <アドレス>で指定されるオフセット値を加えた値となります.

現在のセグメントベースの値をBASICで得る方法はありませんので, ユーザーが責任をもってDEF SEG文によりセグメントベースを指定しておく必要があります.

なお, <アドレス> の値は&H0～&HFFFF (0～65535) の範囲でなければなりません. 小数点以下がある場合は整数化してから実行します.

指定番地へデータを書き込むときにはPOKE文を使います.

参 照 DEF SEG, POKE

サンプルプログラム

```
100 '
110 ' ホームポジションの文字を反転表示する
120 ' このプログラムは、横表示画面で実行してください
130 '
140 WIDTH 80,25
150 LOCATE 1,1:PRINT "H"
160    ' ＢＩＯＳセグメント
170 DEF SEG=&H40
180    ' ホームポジションのオフセットアドレスを読む
190 HOME.X=PEEK(&H600)+PEEK(&H601)*256
200    ' ホームポジションのセグメントアドレスを読む
210 HOME.Y=PEEK(&H602)+PEEK(&H603)*256
220 IF HOME.Y>=32768! THEN HOME.Y=HOME.Y-65536!
230 FOR I=0 TO 19
240 DEF SEG=HOME.Y:POKE HOME.X,PEEK(HOME.X) XOR 255
250 HOME.Y=HOME.Y+8
260 IF HOME.Y>=&HE000 THEN HOME.Y=HOME.Y-&H2000
270 NEXT I
█
Ok
run
```

PLAY
ステートメント

機　能　文字列で表現された音楽を演奏する.

書　式　PLAY 〈文字列〉

文　例　PLAY "MNT150L4O5CDEFGABO6C"

解　説　ミュージックマクロ命令をサポートするステートメントです.
ミュージックマクロ命令は文字列で表される音楽演奏用コマンドで，英数字の組合せで音符を表現するようにしたものです.

次に各コマンドについて説明します.

O 〈式〉 ―――― オクタープを〈式〉の値にセットします. O1～O6 の範囲でセットでき，オクターブの区切りは音名 B と C の間となります. オクターブは一度セットすると再度セットするまで効力があります. O コマンド省略時には O4 にセットされています.

> ―――― オクターブの値を 1 段階高くします. 複数個指定したときは，その個数分だけ高くします.

< ―――― オクターブの値を 1 段階低くします. 複数個指定したときは，その個数分だけ低くします.

| A ⋮ G | + # − | 〔〈式〉〕 ― 音名 A～G（ハ～ロ）の音符を演奏します. 音名の後ろにプラス記号（+）またはナンバー記号（#）をつけるとその音はシャープ（半音上がる）となり，マイナス記号（−）をつけるとその音はフラット（半音下がる）となります. ただし，B+（B #），C−，E+（E #），F−は使用できません.
また〈式〉を指定すると，その音符だけをその長さで演奏します. 長さは L コマンドで指定する値と同じです.〈式〉の値はその音符に対してだけ有効です.〈式〉が省略されたときは，そのとき L コマンドで指定されている値をとります.

N 〈式〉 ―――― 〈式〉で指定される高さの音を演奏します.〈式〉には22～82の範囲の整数を指定します. N22 は O1A，N82 は O6A に対応し，数字が 1 増減すると音は半音上下します.
このコマンドは N＝変数；の形にし，変数に数値データを代入して演奏させるような使用法に向いています. 変数を使ってデータを代入する方法は後述します.

P 〔〈式〉〕 ―――― 休符を表します.〈式〉で指定された長さだけ休みをとります. 長さは L コマンドで指定する値と同じです.〈式〉の値はその休符に対してだけ有効です.

〈式〉が省略されたときは，そのときＬコマンドで指定されている値をとります．

L 〈式〉 ―――― 音符，休符の長さを〈式〉の値にセットします．L1～L64 の範囲でセットできます．各音(休)符の長さは，例えば，４拍子で表すと次のようになります．

Ｌ１　全音(休)符（４拍）
Ｌ２　２分音(休)符（２拍）
Ｌ３　４拍３連音(休)符（４／３拍）
Ｌ４　４分音(休)符（１拍）
Ｌ６　２拍3連音(休)符（２／３拍）
Ｌ８　８分音(休)符（１／２拍）
Ｌ16　16分音(休)符（１／４拍）
Ｌ32　32分音(休)符（１／８拍）
Ｌ64　64分音(休)符（１／16拍）

つまり，各音(休)符の長さは拍子数／〈式〉拍となるわけです．Ｌコマンドは一度セットすると再度セットするまで有効です．Ｌコマンド省略時にはＬ４にセットされています．

.（ピリオド） ―― 付点を表します．音(休)符の後ろに１個つけると，その音(休)符を３／２倍の長さで演奏します．２個つければ(３／２)×(３／２)＝９／４倍，３個つければ27／８倍の長さとなります．音符あるいは休符以外につけることはできません．

T 〈式〉 ―――― テンポを〈式〉の値にセットします．〈式〉には，１分間に演奏する４分音符の数を32～255の範囲で指定します．Ｔコマンド省略時には T120 にセットされています．

ML ―――― ミュージックレガート．各音符は指定された長さで演奏されます．

MN ―――― ミュージックノーマル．各音符は指定された長さの７／８の長さで演奏されます．

MS ―――― ミュージックスタカート，各音符は指定された長さの３／４の長さで演奏されます．

X 〈文字変数〉； ―― 〈文字変数〉中に代入されているミュージックマクロ命令を実行します．〈文字変数〉には，ミュージックマクロ命令の文字列をあらかじめ入れておきます．〈文字変数〉の後のセミコロン（；）は必ず入れてください．

Ｘコマンドは楽譜のサブルーチン的な使い方をすることができます．

ミュージックマクロ命令では，〈式〉にふつう定数を指定しますが〝〈コマンド〉=〈変数名〉;〟の形にすれば，数値変数を指定することもできます．この場合，〈変数名〉の後ろにはセミコロン(;)を必ず入れなければなりません．こうすることにより，〈変数名〉の値をプログラム上で操作でき，めんどうな文字列演算を行うことなくミュージックマクロ命令をコントロールすることができます．

なお，各コマンドの区切りとしてセミコロン（;）を使用してもかまいませんが，ML，MN，MS各コマンドの後ろにはセミコロンを付加しないようにしてください．

サンプルプログラム

```
100 FOR I=1 TO 4
110 READ A$
120 PLAY A$
130 NEXT
140 DATA O2MLL4CL8CDEEL4GL8EEDDL4CP4
150 DATA EL8EFL4GO3CO2AO3CO2GP4
160 DATA O3CCO2BGL8AAAAL4GP4
170 DATA CL8CDEEL4GL8EEDDL4CP4
```

PMAP
関数

機能　スクリーン座標，ワールド座標の相互座標変換を行う．

書式　PMAP（<数式1>，<機能>）

文例　SY1＝PMAP（Wy1，1）

解説　<数式>で指定されるスクリーン座標あるいはワールド座標を，<機能>の指定に従い，対応するワールド座標あるいはスクリーン座標に変換します．

<機能>は変換の種類を指定するもので0から3までの値をとり，それぞれ次のような意味をもちます．

0：<数式>をワールド座標系におけるx座標とみなし，これに対応するスクリーン座標系でのx座標に変換します．(Wx → Sx)

1：<数式>をワールド座標系におけるy座標とみなし，これに対応するスクリーン座標系でのy座標に変換します．(Wy → Sy)

2：<数式>をスクリーン座標系におけるx座標とみなし，これに対応するワールド座標系でのx座標に変換します．(Sx → Wx)

3：<数式>をスクリーン座標系におけるy座標とみなし，これに対応するワールド座標系でのy座標に変換します．(Sy → Wy)

PMAP関数を用いれば，ワールド座標からスクリーン座標の変換ができますから，GET@文やPUT@文のように，スクリーン座標で位置を指定する操作の場合にもワールド座標を変換して指定することができます．なお，スクリーン座標系の原点は，左上隅（0，0）です．

注意：PMAP関数は，変換の結果がスクリーン座標系あるいはワールド座標系から外れてもエラーになりませんので注意してください．

参照　文法編　ウィンドウとビューポート

サンプルプログラム

```
100 '** World <--> screen **
110 SCREEN 1
120  WINDOW(-100,-100)-(100,100)
130  VIEW(0,0)-(719,511)
140  PRINT TAB(5);"window(-100,-100)-(100,100)"
150  PRINT TAB(5);"view(0,0)-(716,511)"
160  PRINT "ﾆ ｵｲﾃ World <--> Screen ﾉ ﾍﾝｶﾝｦ ｼﾏｽ｡"
170  PRINT
180 *ENTRY
190  PRINT  "World --> Screen : 1"
200  PRINT  "Screen --> world : 2"
210  PRINT  "---------->  END  : 3"
```

```
220  INPUT ANS
230  IF ANS=3 THEN *OWARI ELSE IF ANS<>1 AND ANS<>2 THE
N *ENTRY
240  INPUT "Coordinate =";CO
250  IF ANS=2 THEN *WFROMS
260  PRINT "WX :";CO,"SX =";PMAP(CO,0)
270  PRINT "WY :";CO,"SY =";PMAP(CO,1)
280  GOTO *ENTRY
290 *WFROMS
300  PRINT "SX :";CO,"WX =";PMAP(CO,2)
310  PRINT "SY :";CO,"WY =";PMAP(CO,3)
320 GOTO *ENTRY
330 *OWARI
340  WINDOW:VIEW:END
```

```
run
    window(-100,-100)-(100,100)
    view(0,0)-(716,511)
ｺﾆ ｵｲﾃ World <--> Screen ﾉ ﾍﾝｶﾝｦ ｼﾏｽ｡

World --> Screen : 1
Screen --> world : 2
---------> END   : 3
? 1
Coordinate =? 0
WX : 0          SX = 360
WY : 0          SY = 255
World --> Screen : 1
Screen --> world : 2
---------> END   : 3
? 2
Coordinate =? 100
SX : 100        WX =-72.1836
SY : 100        WY = 60.86
World --> Screen : 1
Screen --> world : 2
---------> END   : 3
? 3
```

POINT
ステートメント

機能	LP（最終参照点）を変更する.

書式

POINT ｛(Wx, Wy) / STEP (x, y)｝

文例

POINT (100, 150)

解説

最後にグラフィック操作の行われた座標（LP）を，（Wx，Wy）で指定されるワールド座標上の点に変更します．LP は相対座標によって座標指定を行う際の基点となりますが，POINT 文を使用すればこの LP を指定した点に設定することができます．したがって DRAW 文の描画開始点をセットしたり，LINE 文で始点をセットしたりすることができます．
例えば，

LINE (100, 100)－(200, 150), 5

は，次のようにしても同じ結果を得ることができます．

POINT (100, 100)
LINE －STEP (100, 50), 5

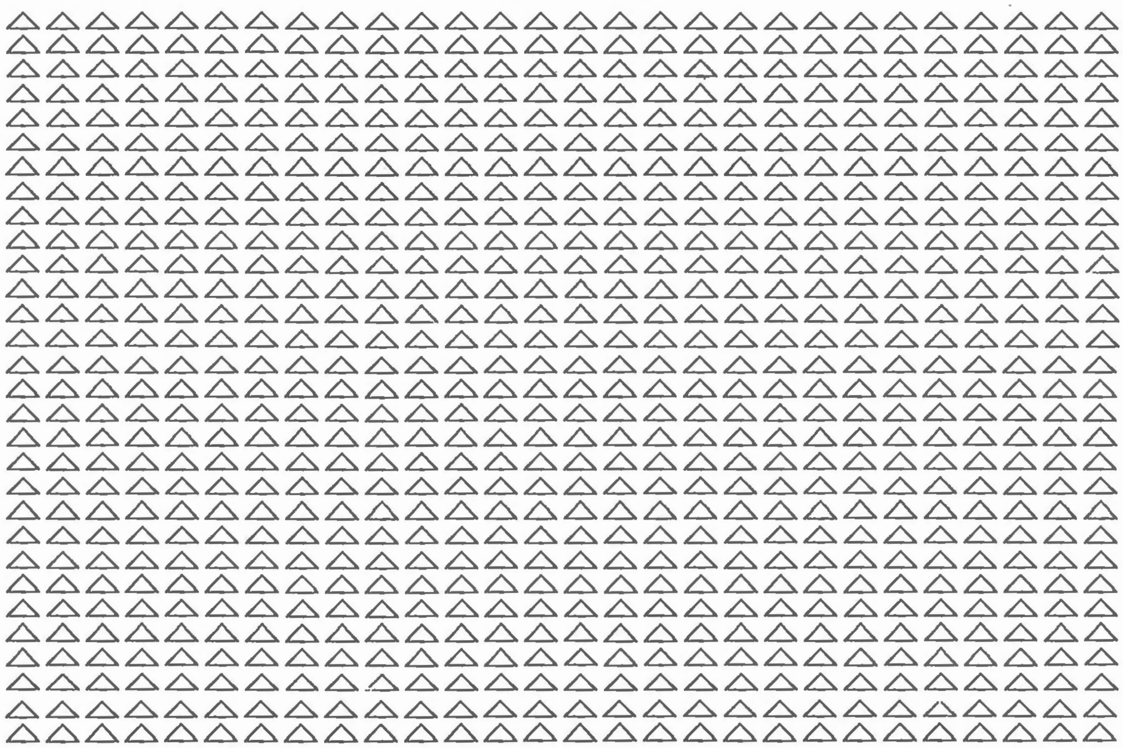
POINT 文に STEP オプションをつけたときは，相対座標による指定となります．なお，現在の LP を得るには POINT 関数を用います．

参照

POINT（関数）

サンプルプログラム

```
100 CLS:LOCATE ,,0
110 POINT (0,0)
120  FOR I=1 TO 30
130   FOR J=1 TO 28
140    POINT STEP(25,0)
150       GOSUB SANKAKU
160    NEXT
170    POINT STEP(-700,15)
180  NEXT
190 END
200 SANKAKU
210       CO=RND*14+1
220       LINE -STEP(-10,10),CO
230       LINE -STEP(20,0),CO
240       LINE -STEP(-10,-10),CO
250 RETURN
```



POINT
関数

機 能

1）LP（最終参照点）の値を返す．

2）スクリーン座標の指定された点に表示されているピクセルの色を返す．

書 式

1）POINT（〈機能〉）

2）POINT（Sx，Sy）

文 例

1）LX＝POINT（0）

2）C＝POINT（250，300）

解 説

POINT 関数は，指定するパラメータにより，異なった働きをします．

1）〈機能〉が指定されたときは LP（最終参照点）を返します．

2）Sx，Sy が指定されたときはその点のピクセルの色を返します．

次に，それぞれの場合について説明します．

1）〈機能〉指定時

最後にグラフィック操作の行われた座標（LP）を〈機能〉の指定により，スクリーン座標あるいは，ワールド座標で返します．

〈機能〉は 0 から 3 の整数値をとり，それぞれの場合に POINT 関数の返す値の意味は次のとおりです．

0：最後にグラフィック操作の行われた点の x 座標を，ワールド座標系上の値に変換して返します．

1：同じく y 座標を返します．

2：最後にグラフィック操作の行われた点の x 座標を，スクリーン座標系上の値に変換して返します．

3：同じく y 座標を返します．

最後に操作の行われた座標がウィンドウの外であった場合には，POINT 関数の返す座標に対応する点は，画面上には表示されていない点であることに注意してください．またこの時，スクリーン座標系に変換されて返された値である場合には，ビューポート外になることに注意してください．

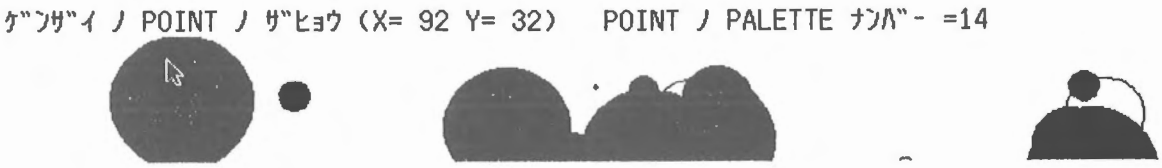
2）Sx，Sy 指定時

(Sx，Sy）により指定されたスクリーン座標上に表示されているピクセルの色をパレット番号で返します．指定された座標（Sx，Sy）がビューポートの外にある場合には，POINT 関数は－1を返します．

座標はスクリーン座標で指定することに注意してください．白黒モードの場合には，POINT 関数は，リセット（黒＝0）またはセット（白＝15）をピクセルの色として返します．

サンプルプログラム

```
100 MOUSE 0:SCREEN 1:CLS :LOCATE 1,1,0
110 FOR I=1 TO 100
120 X=RND*620+50:Y=RND*411+50:R=RND*50:PA=RND*15:PA1=RN
D*15
130 CIRCLE(X,Y),R,PA
140 PAINT(X,Y),PA1,PA
150 NEXT
160 MOUSE 1,0,0,1
170 X=MOUSE(0)-2:Y=MOUSE(1)-2
180 CO=POINT (X,Y)
190 LOCATE 1:PRINT USING"ｹﾞﾝｻﾞｲ ﾉ POINT ﾉ ｻﾞﾋｮｳ (X=###"
;X;:PRINT USING" Y=###";Y;:PRINT USING")   POINT ﾉ PALE
TTE ﾅﾝﾊﾞｰ =##";CO
200 GOTO 170
```



POKE
ステートメント

機能 メモリ上の指定番地へデータを書き込む.

書式 POKE <アドレス>, <式>

文例 POKE &HF000, &HFF

解説 <アドレス>で指定されるメモリ上のオフセットアドレスに1バイト（8ビット）のデータを書き込みます.

メモリ上の実際のアドレスは，最後に実行されたDEF SEG文によって指定されたセグメントベースに，<アドレス> で指定されたオフセット値を加えた値となります.

<アドレス>の値は，&H0～&HFFFF（0～65535）の範囲でなければなりません．小数点以下がある場合は整数化してから実行します.

<式>は書き込まれるデータで，&H0～&HFF(0～255)の値でなければなりません．もし小数点以下があると整数化してから実行されます.

ただしこのステートメントは，現在のメモリの内容を書き換えてしまうため，不用意に使うとBASICが使用している作業領域を壊してしまい，誤動作の原因となることもあります．使うときには，使用可能な領域かどうかを確認してください.

また，現在のセグメントベースの値をBASICで得る方法はありませんので，ユーザーが責任をもってDEF SEG文によりセグメントベースを指定しておく必要があります.

POKE文の逆の働きをする関数としてPEEK関数が用意されています．POKE文とPEEK関数は，効率的なデータの格納や機械語サブルーチンとの引数の受渡しなどに役立ちます.

参照 DEF SEG, PEEK

サンプルプログラム

```
100 '
110 ' ホームポジションの文字を反転表示する
120 ' このプログラムは、横表示画面で実行してください
130 '
140 WIDTH 80,25
150 LOCATE 1,1:PRINT "H"
160   ' ＢＩＯＳセグメント
170 DEF SEG=&H40
180   ' ホームポジションのオフセットアドレスを読む
190 HOME.X=PEEK(&H600)+PEEK(&H601)*256
200   ' ホームポジションのセグメントアドレスを読む
210 HOME.Y=PEEK(&H602)+PEEK(&H603)*256
220 IF HOME.Y>=32768! THEN HOME.Y=HOME.Y-65536!
```

```
230 FOR I=0 TO 19
240 DEF SEG=HOME.Y:POKE HOME.X,PEEK(HOME.X) XOR 255
250 HOME.Y=HOME.Y+8
260 IF HOME.Y>=&HE000 THEN HOME.Y=HOME.Y-&H2000
270 NEXT I
```

```
Ok
```

POS
関数

機能 現在のカーソルの桁位置を返す.

書式 POS (<ダミー引数>)

文例 P=POS (0)

解説 <ダミー引数>には，数値であれば何を指定してもかまいません.

現在のカーソルの桁位置（水平方向）を返します.

返される値の範囲は，1から，WIDTH 文によって指定された最大の桁数までで，値が小さいほど画面の左の方を意味します.

なお，カーソルの行位置（垂直方向）を知るためには CSRLIN 関数を使用します.

参照 CSRLIN

サンプルプログラム

```
100 A=RND*25+65
110 IF POS(0)>30 AND POS(0)<50 THEN COLOR 13:A=A+32 ELSE
COLOR 15
120 PRINT CHR$(A);
130 GOTO 100
```

```
run
WDCBSJGGJBLYNYCIXMKVCBTWYPXIDGyiymdoliwsvurxuulrrBNECJBLWFFGAKATAFISNOYBCNFEJEME
FLISMQNSVUSRSIYSRIQUBPYYTGYBHNqpetrwmnpnpklvkofbqVZCGXFHMBVVTNISGYMWEMQOSDFCGREL
NUHWGDEUXWSUZPXJDERZSBDSWWXPJEgtzgjkesfgsvogcivhkJENIUJRTUOSYNVSSEFYFFTLPYFKEGMG
SOVSDFUHAVYTDIJDJQRWNEFEEFPOLVhbuupwguwhhccxbdguoRRKHCZEBRQCCRPMHEJIWEYNGNUOXKWL
YIEUPNNHYKGEFIWHUPAHFUMRNGAWVSrwgpzjqzdfnjicyfniaFMBHGVPUGQGYPAXBNCKPBIULPTENYXH
PYRWJ
```

PRESET

ステートメント

機 能	指定された座標に描かれた点を消去する.
書 式	PRESET { (Wx, Wy) / STEP (x, y) } 〔, パレット番号〕
文 例	PRESET (100, 100)

解 説

ワールド座標の (Wx, Wy) で表される位置に描かれた点を消去します. STEP を付けた場合は相対座標による指定となります (文法編 座標の指定の仕方参照).

このステートメントは, オプションの 〈パレット番号〉 を付けない方が一般的な使い方で, その場合, 現在 COLOR 文によって設定されているバックグラウンドカラーで点を塗り変えます.

〈パレット番号〉を指定した場合は, PSET 文と全く同じに機能し, そのパレット番号のカラーで点を描きます. PRESET は LP を指定された座標 (Wx, Wy) に設定します.

参 照

PSET, COLOR

サンプルプログラム

```
100 SCREEN 1:CLS
110     POINT (0,100)
120 FOR I=0 TO 200
130   PSET STEP(3,0),I MOD 15:NEXT I
140     POINT(0,100)
150 FOR I=0 TO 200
160   PRESET STEP(3,0)
170 NEXT I
```

PRINT
ステートメント

機　能　ディスプレイ画面にデータを出力する.

書　式　PRINT 〔〈式〉 | ; | ・・・〕
　　　　　　　　　　　 | , |

文　例　PRINT "ABC"

解　説　〈式〉で表される数値変数，文字変数，数値定数，文字定数などのデータをディスプレイ画面に出力します.

〈式〉が省略されている場合は改行のみを行います.

式の値や文字列を表示する領域は，あらかじめ各行が15文字ごとに分割して定められていて，〈式〉をカンマ(,)で区切って並べると，データは各行の各領域の先頭からそれぞれ表示されます．したがって，表示の頭ぞろえを行うことができます．〈式〉の最後にカンマをつけた場合，次に実行されるPRINT／LPRINT文で出力されるデータは次の領域に表示されます.

また〈式〉をセミコロン（；）またはスペースで区切って並べると，データは間隔なしに詰めて表示されます．〈式〉の最後にセミコロンをつけた場合は，データの表示後に改行が行われず，次に実行されるPRINT／LPRINT文で出力されるデータは，その直後に表示されます.

数値を表示した場合その後ろに必ず空白が1つ挿入されます．また，数値の前に正負記号のための桁を確保します.

単精度の数値で，指数（浮動小数点）形式でなくても6桁以下の桁数で精度に影響を及ぼさず表示できるものは，固定小数点形式で表示されます.

同様に倍精度の数値は14桁以下の桁数で精度に影響を及ぼさず表示できるものは，固定小数点形式の表示になります.

このステートメントのキーワード"PRINT"の簡略形としてクェスチョンマーク（？）を使用することもできます.

表示する文字列や数値の長さが現在のカーソルの位置より後方にとれず，それを表示すると次の行に渡ってしまうような場合には改行してそれを表示します.

参　照　PRINT USING，LPRINT／LPRINT USING

サンプルプログラム

```
100 SCREEN 0,0,1
110 FOR I=2 TO 15
120  FOR J=0 TO 15
130   PRINT CHR$(I*16+J);SPC(3);
140  NEXT J:PRINT
150 NEXT I
160 END
```

```
run
    !   "   #   $   %   &   '   (   )   *   +   ,   -   .   /
0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ?
@   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
P   Q   R   S   T   U   V   W   X   Y   Z   [   ¥   ]   ^   _
`   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o
p   q   r   s   t   u   v   w   x   y   z   {   ¦   }   ~   ▒
▔   ▂   ▃   ▄   ▅   ▆   ▇   █   ▏   ▎   ▍   ▌   ▋   ▊   ▉   ┼
┴   ┬   ┤   ├   ▔   ─   │   ▕   ┌   ┐   └   ┘   ╭   ╮   ╰   ╯
    。  「  」  、  ・  ヲ  ァ  ィ  ゥ  ェ  ォ  ャ  ュ  ョ  ッ
ー  ア  イ  ウ  エ  オ  カ  キ  ク  ケ  コ  サ  シ  ス  セ  ソ
タ  チ  ツ  テ  ト  ナ  ニ  ヌ  ネ  ノ  ハ  ヒ  フ  ヘ  ホ  マ
ミ  ム  メ  モ  ヤ  ユ  ヨ  ラ  リ  ル  レ  ロ  ワ  ン  ゛  ゜
═   ╞   ╪   ╡   ◢   ◣   ◥   ◤   ♠   ♥   ♦   ♣   ●   ○   ╱   ╲
╳   円  年  月  日  時  分  秒
Ok
```

PRINT USING

ステートメント

機能　ディスプレイ画面にデータを指定した書式で出力する.

書式　PRINT USING 〈書式制御文字列〉〔 | ; / , | 〔〈式〉 | ; / , | ・・・〕〕

文例　PRINT USING "#####,." ; A, B

解説　〈式〉で表される数値変数，文字変数，数値定数，文字定数などのデータを，〈書式制御文字列〉によって指定された書式でディスプレイ画面に出力します.

文字列の書式制御

! ―――― 与えられた文字変数や文字定数の最初の1バイトだけ出力します.

&〈n個の␣（スペース）〉&

与えられた文字変数や文字定数の先頭から（n＋2）バイトの文字列を出力します. 与えられた文字列が（n＋2）バイトより長い場合は，余分な文字は無視され，短い場合には，文字列は左づめに出力され，残った部分にはスペースが出力されます.

@ ―――― 与えられた文字変数や文字定数の文字列全体を出力します. 複数個指定した場合は1つの"@"に対して〈式〉中の1つの文字列が出力されます. "@"の数が〈式〉の数より多い場合，残りの"@"は無視されます.

数値の書式制御

―――― 数値を出力する桁数を指定します. 指定した桁数より数値の桁数が小さいときには，右づめで出力され，残った場合にはスペースが出力されます.

. ―――― 小数点の位置を指定します. 小数点以下の桁は，その桁の値が0であっても必ず表示されます.

＋ ―――― 〈書式制御文字列〉の最初または最後につけた場合，数値の正負記号（プラス，マイナス）がそれぞれ前または後に出力されます. 2個以上の"＋"を並べた場合，余分は制御文字以外の文字と同じ扱いになります.

－ ―――― 〈書式制御文字列〉の最後につけた場合，数値が負の時に数値の後ろに"－"が出力されます. 前につけたり，2個以上並べた場合，余分は制御文字以外の文字と同じ扱いになります.

＊＊ ――― 〈書式制御文字列〉の先頭につけた場合，数値領域の左側にスペースがあるとそこを"＊"で埋めて出力します. この"＊＊"は2桁分の領域を確保します.

¥¥ ――― 〈書式制御文字列〉の先頭につけた場合，出力される数値の直前に"¥"を出力します. "¥¥"は2桁分の領域を確保しますが，このうち1桁分は"¥"の出力領域とし

て使われます．指数形式の書式指定と同時に使用することはできません．

＊＊¥ ―― 〈書式制御文字列〉の先頭につけた場合，出力される数値の直前に〝¥″を出力し，さらにその前にスペースがあるとアスタリスク〝＊″を埋めて出力します．〝＊＊¥″は3桁分の領域を確保しますが，このうち1桁分は〝¥″の出力領域として使われます．

， ―――― 桁数指定の〝#″の並びの中においた場合，数値の整数部が3桁ごとに〝，″で区切って出力します．ただし，上記の〝．″より右側に置いた場合は，数値の最後に〝，″が出力され，3桁ごとの区切りは行われません．指数形式の書式指定と同時に使用することはできません．

^^^^ ― 桁数指定の〝#″の後につけた場合，数値は指数形式で出力されます．〝^^^^^″は4桁分の領域を確保し，E±nn の表示に当てます．

_ ―――― 上記の制御文字1文字を単に文字として表示するために使用します．〝_″に続く1文字は常に書式制御機能を持たない文字として出力されます．

制御文字以外の文字

以上の制御文字以外の文字を置いた場合，数値の前や後ろにそのキャラクタが出力されます．

数値の領域を越えた場合

指定した数値領域より数値の桁数が大きい場合，数値の直前に〝%″が出力されます．数値が丸めによって領域より大きくなった場合も，丸めた数値の前に〝%″が出力されます．

〈書式制御文字列〉中の各制御文字の数より〈式〉の数の方が多い場合

この場合，〈式〉の並びに対して〈書式制御文字列〉全体が繰り返し使われます．

```
PRINT USING "@##␣"; "JAPAN"; 90; "USA"; 80; "USSR"; 3
JAPAN90 USA80 USSR3
Ok
```

参照　PRINT，LPRINT／LPRINT USING

サンプルプログラム

```
100 A=123.456
110 PRINT USING "!!";"NEC PC-100"
120 PRINT USING "&      &";"NEC PC-100"
130 PRINT USING "#####";A
140 PRINT USING "#####.#";A
150 PRINT USING "+####.#";-A
160 PRINT USING "#####.#-";-A
170 PRINT USING "**###.#";A
180 PRINT USING "¥¥###.#";A
190 PRINT USING "**¥##.#";A
200 PRINT USING "#####,.#";A*10
210 PRINT USING "#####,.#^^^^";A*10
220 PRINT USING "¥¥###._-";A
230 PRINT USING "#####";123456!
240 PRINT USING "@ is a @.";"This";"PEN";"That","DESK"
250 END

run
N
NEC PC-1
  123
  123.5
 -123.5
  123.5-
**123.5
 ¥123.5
*¥123.5
 1,234.6
 12345.6E-01
 ¥123.-
%123456
This is a PEN.That is a DESK.
Ok
```

PRINT#
ステートメント

機能　シーケンシャルファイルにデータを書き出す.

書式　PRINT #〈ファイル番号〉,〔〈式〉・・・〕

文例　PRINT #2, A, B, C

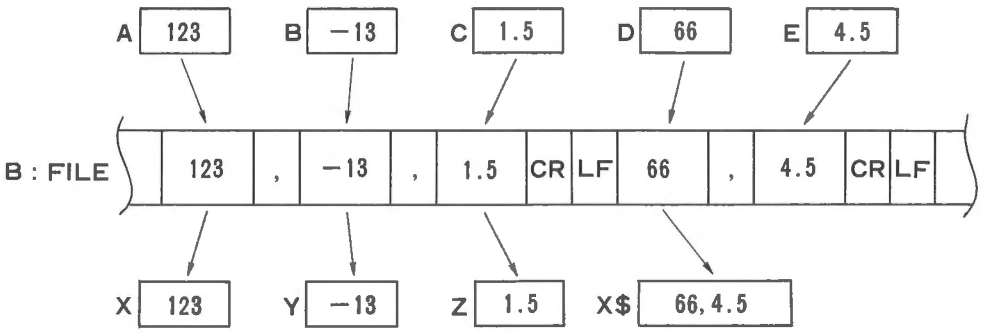
解説　あらかじめOPEN文で出力モードを指定して開いておいたファイルへ,〈式〉で表す数値や文字列などのデータを書き出します.PRINT #文はデータを書き出すファイルが"SCRN:"(ディスプレイ画面)であればPRINT文と同じ働きをし,"LPT1:"(プリンタ)であればLPRINT文と同じ働きをします.また,ディスクファイルへデータを書き出した場合には,INPUT #文,LINE INPUT #文,INPUT$関数を使ってそのディスクファイルからデータを読み込むことができます.

〈ファイル番号〉は,OPEN文でシーケンシャルファイルを開くときに指定した番号です.

〈式〉は,ファイルに書き出す数値,または文字列などのデータです.複数の〈式〉を書き出す場合は〈式〉と〈式〉の間をセミコロン(;)かカンマ(,)で区切ります.

〈式〉が数値のときは,PRINT #文を実行すると,文字列に変換された数値がカンマで区切って出力され,最後にキャリッジリターンコード(CHR$(13))+ラインフィードコード(CHR$(10))が出力されます.

```
OPEN "B:FILE" FOR OUTPUT AS#1
PRINT #1,A;B;C
PRINT #1,D;E
CLOSE #1
```




```
OPEN "B:FILE" FOR INPUT AS#1
INPUT #1,X,Y,Z
LINE INPUT #1,X$
CLOSE #1
```

このようにしてディスク上にいったんファイルを作成しておくと,INPUT #文を使ってカンマで区切られたデータを読み込んだり,LINE INPUT #文を使ってキャリジリターンコード+ラインフィードコードごと(すなわち1行ごと)に読み込んだり,また,INPUT$関数を使って指定した長さだけのデータを読み込んだりすることができます.

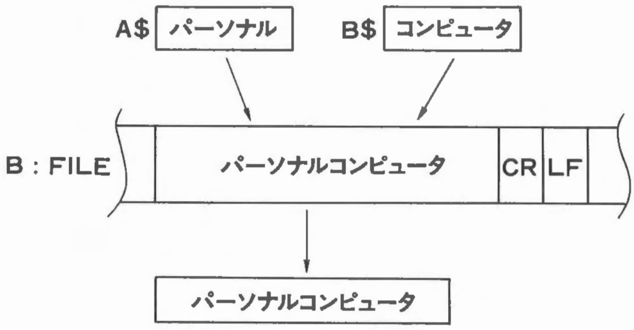
PRINT

しかしPRINT #文で書き出すデータは，PRINT文で画面へ表示する内容と同じですから，ディスクファイルへ書き出す場合，区切り記号にカンマ使うと余分な空白がファイルに書き出されてしまいます．

〈式〉の内容が文字列のときは，PRINT #文を実行すると〈式〉で指定した文字列が連続してファイルに書き出され，最後にキャリッジリターンコード（CHR$(13)）＋ラインフィードコード（CHR$(10)）が書き出されます．

〈式〉と〈式〉とをセミコロンで区切ると，数値の場合と異なり，データはカンマで区切られずにディスクファイルへ書き込まれます．このため，INPUT #文でこれらのデータを入力するときに別々の文字列として読み込むことはできません．

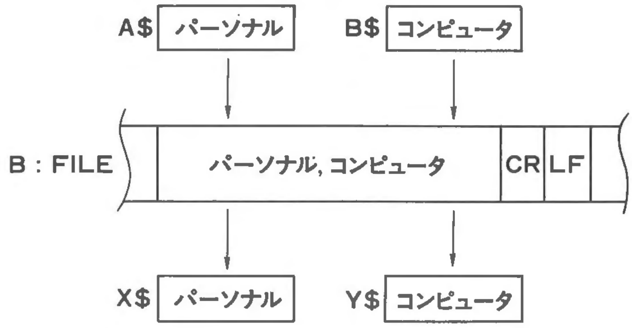
```
OPEN  "B:FILE" FOR OUTPUT AS#1
PRINT  #1,A$;B$
CLOSE  #1
```




```
OPEN  "B:FILE" FOR INPUT AS#1
INPUT  #1,X$
CLOSE  #1
```

INPUT #文で2つの〈式〉を別々の文字列として読み込みたい場合は，PRINT #文中の〈式〉と〈式〉の間に，区切り記号としてカンマを積極的に入れてください．

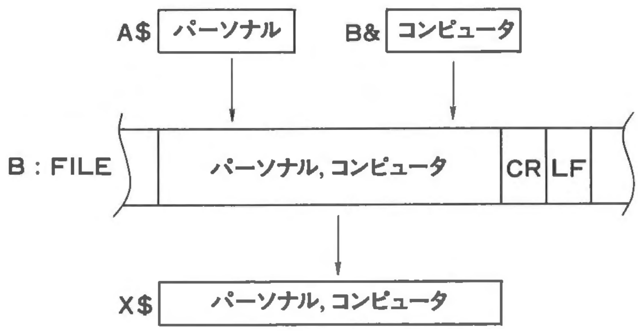
```
OPEN  "B:FILE" FOR OUTPUT AS#1
PRINT  #1, A$;",";B$
CLOSE  #1
```




```
OPEN  "B:FILE" FOR INPUT AS#1
INPUT  #1, X$, Y$
CLOSE  #1
```

LINE INPUT #文を使うと，キャリッジリターンコード＋ラインフィードコードを区切りとしてデータが読み込まれます．

```
OPEN  "B:FILE" FOR OUTPUT AS#1
PRINT  #1, A$;",";B$
CLOSE  #1
```

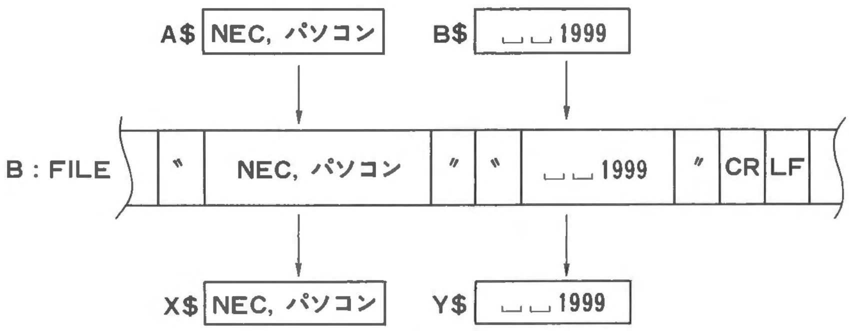



```
OPEN  "B:FILE" FOR INPUT AS#1
LINE INPUT  #1, X$
CLOSE  #1
```

文字列中にカンマ，セミコロン，キャリッジリターンコード＋ラインフィードコードを含む場合，または文字列の先頭に意味のある空白がある場合は，PRINT #文中の〈式〉の前後を独立したダブルクォーテーション（CHR$(34)）で囲むようにしてください．

PRINT

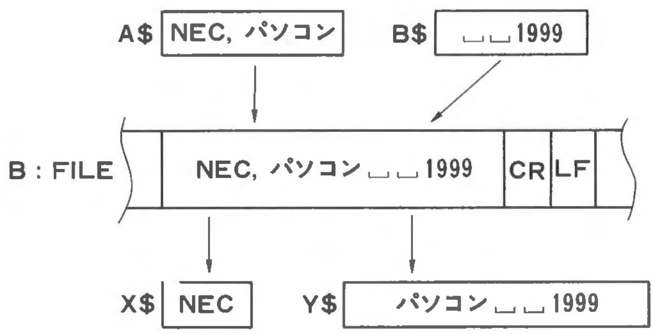
```
OPEN "B:FILE" FOR OUTPUT AS#1
PRINT #1, CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)
CLOSE #1
```




```
OPEN "B:FILE" FOR INPUT AS#1
INPUT #1, X$, Y$
CLOSE #1
```

上記の例で，もし〈式〉の前後をダブルクォーテーションで囲まないと，次のような結果となってしまいます．

```
OPEN "B:FILE" FOR OUTPUT AS#1
PRINT #1, A$;B$
CLOSE #1
```




```
OPEN "B:FILE" FOR INPUT AS#1
INPUT #1, X$, Y$
CLOSE #1
```

PRINT USING 文と同じように，PRINT # USING 文を使用してファイルに書き出すデータの書式を指定することもできます．

参照　OPEN，CLOSE，PRINT # USING，INPUT #，LINE INPUT #

サンプルプログラム

```
100 A$="abcde":B$="fghij":C$="klmno"
110 OPEN "test10" FOR OUTPUT AS #1
120 PRINT#1,A$;B$;C$
130 PRINT#1,A$,B$,C$
140 CLOSE
150 OPEN "test10" FOR INPUT AS #1
160 LINE INPUT#1,X$
170 LINE INPUT#1,Z$
180 CLOSE
190 PRINT X$
200 PRINT Z$
```

```
run
abcdefghijklmno
abcde           fghij           klmno
Ok
```

PRINT# USING

ステートメント

機能	シーケンシャルファイルにデータを指定した書式で出力する.
書式	PRINT # <ファイル番号>, USING <書式制御文字列> ; <式> 〔 ; / , <式> ; / , ・・・〕
文例	PRINT #2, USING "####" ; A

解説

〈ファイル番号〉で指定されたシーケンシャルファイルに対して, 〈式〉で表される文字列や数値などのデータを書式指定して出力します.

〈ファイル番号〉は, そのシーケンシャルファイルが出力モードとしてオープンされたときに指定された番号です.

PRINT # USING 文は, その出力先がシーケンシャルファイルであることを除けば, その機能は PRINT USING 文と同じです.

参照 PRINT USING, PRINT #, OPEN

サンプルプログラム

```
100 OPEN "test11" FOR OUTPUT AS #1
110 FOR I=1 TO 15 STEP .5
120 X=I^2
130 PRINT #1,X
140 PRINT #1,USING "##.#";X
150 NEXT
160 CLOSE
170 OPEN "test11" FOR INPUT AS #1
180 FOR I=1 TO 15 STEP .5
190 INPUT #1,A$
200 INPUT #1,B$
210 PRINT A$,:PRINT B$
220 NEXT
230 CLOSE

run
1              1.0
2.25           2.3
4              4.0
6.25           6.3
9              9.0
12.25          12.3
```

```
16          16.0
20.25       20.3
25          25.0
30.25       30.3
36          36.0
42.25       42.3
49          49.0
56.25       56.3
64          64.0
72.25       72.3
```

PSET

ステートメント

機　能　指定された座標に点を描く．

書　式　PSET ｜(Wx, Wy)｜〔, 〈パレット番号〉〕
　　　　　　　　 ｜STEP (x, y)｜

文　例　PSET (100, 100), 4

解　説　ワールド座標の（Wx, Wy）で表される位置に点を描きます．STEP をつけた場合は相対座標による指定となります（文法編　座標の指定の仕方参照）．

〈パレット番号〉には点の色を指定しますが，もし省略された場合は，現在のフォアグラウンドカラー（COLOR 文参照）が採用されます．

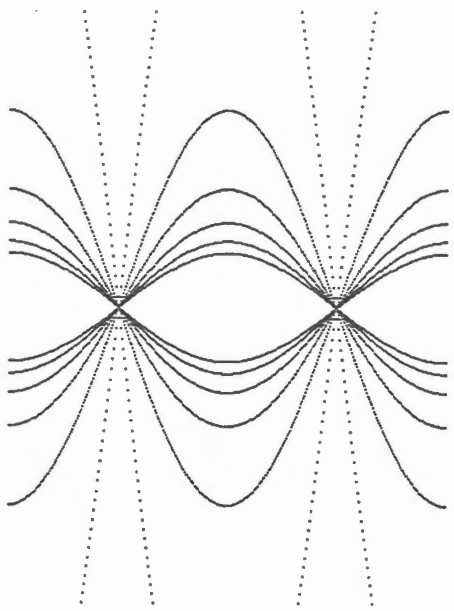
PSET は LP を指定された座標（Wx, Wy）に設定します．

参　照　COLOR, PRESET

サンプルプログラム

```
100 CLS :LOCATE 1,1,0
110 FOR J=-5.5 TO 5.5 STEP 1
120 X=250
130 FOR I=0 TO 360
140 CO=RND*14+1
150 S=COS(I/180*3.14159)
160 PSET(X,256-255*S/J),CO
170 X=X+1:NEXT:NEXT
180 BEEP
190 PRINT "HIT ANY KEY TO END"
200 IF INKEY$="" THEN 200
210 END
```

HIT ANY KEY TO END



PUT
ステートメント

機　能　バッファ中のデータをファイルに書き出す．

書　式　PUT〔#〕〈ファイル番号〉〔,〈整数〉〕

文　例　PUT3, 5

解　説　〈ファイル番号〉で指定されたファイルに対応するバッファの内容を書き出します．PUT は，指定されたファイルがディスクファイルであるかどうかで，その動作が異なります．

〈ファイル番号〉で指定されたファイルがディスクファイルである場合には，PUT は，ランダムファイルへの書込みとして働きます．指定されたファイルは，ランダムファイルのモードでオープンされていなければなりません．

〈整数〉はファイルのレコード番号として解釈され，バッファ中のデータが指定されたレコードに書き込まれます．〈整数〉が省略された場合には，直前に GET，PUT が行われたレコードの次のレコードに書き込まれます．

〈ファイル番号〉で指定されたファイルがディスクファイルでない場合には，PUT は，バッファ中のデータを，指定されたファイルに対応する装置に対して出力する働きをします．このとき，指定されたファイルは出力モードでオープンされていなければなりません．

〈整数〉はバッファからファイルに対して書き出すバイト数を意味し，0 からファイルバッファサイズまでの値で指定します．〈整数〉が省略されたとき，および 0 が指定された時には，ファイルバッファサイズ分のデータを書き出します．

なお書き出すデータはあらかじめ FIELD 文，LSET／RSET 文により準備しておかねばなりません．

参　照　OPEN，FIELD，GET，LSET／RSET

サンプルプログラム

```
100 '--RANDOM DATA WRITE --
110 OPEN "DATA1" AS #1
120 FIELD #1.30 AS A$.20 AS B$.50 AS C$
130 *ENTRY
140 LINE INPUT "NAME ?";NAM$
150 IF NAM$="END" THEN *EXIT
160 LINE INPUT "TEL ?";TEL$
170 LINE INPUT "ADDRESS?":ADR$
180 LSET A$=NAM$
190 LSET B$=TEL$
200 LSET C$=ADR$
210 PUT #1
220 GOTO *ENTRY
230 *EXIT
240 CLOSE
```

```
run
NAME ?相沢
TEL ?03-001-3418
ADDRESS?東京都　世田谷区　１－１－５
NAME ?岡野
TEL ?00-087-8734
ADDRESS?神奈川県　横浜市　１－２－３
NAME ?END
Ok
```

PUT@

ステートメント

機 能　グラフィックパターン，文字列，漢字を指定座標に表示する．

書 式

1）PUT〔@〕(Sx, Sy)，〈配列変数名〉〔(〈要素〉)〕〔，〈機能〉〕

2）PUT〔@〕(Sx, Sy)，〈文字列〉〔，〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉〕

3）PUT〔@〕(Sx, Sy)，KANJI (〈漢字コード〉)〔，〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉〕

文 例

1）PUT (100, 100)，G%

2）PUT (400, 100)，A$

3）PUT (0, 0)，KANJI (12321)

解 説

1）GET@ステートメントによって配列に読み込まれたグラフィックパターンを画面上の任意の位置に表示します．

座標(Sx, Sy)はGET@ステートメントの場合と同様，スクリーン座標で指定します．表示されるグラフィックパターンは，この(Sx, Sy)で示される点にパターンの左上の頂点を合わせて出力されることになります．このステートメントを実行するとLPは(Sx, Sy)に移動します．

〈配列変数名〉は表示したいグラフィックパターンが格納されている数値型の配列変数の名前であり，〈要素〉は配列内のどの要素番号からデータを取り始めるかの指定です．〈要素〉が省略された場合は，配列の最初から取り始めます．これらの指定は，GET@命令で使ったものと同じものを指定します．

〈機能〉とは，グラフィックパターンを画面に表示する際，いろいろな機能を指定できるもので，以下のものが用意されています．

PSET	配列内のパターンをそのまま表示します．
PRESET	配列内のパターンの各ピクセルのパレット番号を，15－(そのピクセルのパレット番号)におきかえて表示します．
OR	配列内のパターンと，既にある画面上のグラフィックパターンとのOR(論理和)を各ピクセルごとにとり，その結果を画面に表示します．
AND	配列内のパターンと画面上のパターンとのAND(論理積)を各ピクセルごとにとり，その結果を画面上に表示します．
XOR	配列内のパターンと画面上のパターンとのXOR(排他的論理和)を各ピクセルごとにとり，その結果を画面に表示します．

なお，〈機能〉の省略時には XOR が指定されたのと同じになります．

2）〈文字列〉を画面上の任意の位置に表示します．
座標（Sx，Sy）はスクリーン座標で指定しますが，表示される〈文字列〉はその先頭の文字フォントの左上の点がここに合わせられて出力されることになります．

〈文字列〉は文字変数でも文字定数でもよく，2 バイトキャラクタ（漢字など）を含んでいてもかまいません．

〈フォアグラウンドカラー〉および〈バックグラウンドカラー〉は，表示する〈文字列〉の色およびその文字列が出力される背景の色をそれぞれ指定するもので，両方ともパレット番号で指定します．白黒モードのときは，0 を指定すると黒，0 以外を指定すると白で表示されます．

この文は，出力される〈文字列〉をグラフィックパターンとして扱いますので，キャラクタ座標系に関係なく，画面上のどこにでも表示させることができます．しかし，PUT@ステートメントを用いて出力した文字は，スクリーンエディタにより編集することができませんので注意してください．

3）〈漢字コード〉で指定された 2 バイトキャラクタを画面上の任意の位置に表示します．

座標（Sx，Sy）はスクリーン座標で指定しますが，表示される〈漢字コード〉はその文字フォントの左上の点がここに合わせられて出力されることになります．

〈漢字コード〉は内部連続コードにより指定します．
このとき &J または&K（JIS コードまたは JIS 区点コードをそれぞれ内部連続コードに変換するための接頭辞）を使用することもできます（付録　漢字コード表参照）．

〈フォアグラウンドカラー〉および〈バックグラウンドカラー〉は，表示する〈漢字コード〉の色，およびその漢字が出力される背景の色をそれぞれ指定するもので，両方ともパレット番号で指定します．白黒モードのときは，0 を指定すると黒，0 以外を指定すると白で表示されます．

この文は，出力される〈漢字コード〉をグラフィックパターンとして扱いますので，キャラクタ座標系に関係なく，画面上のどこにでも表示させることができます．しかし，PUT@ステートメントを用いて出力した文字は，スクリーンエディタにより編集することができませんので注意してください．

なお，この文では，〈漢字コード〉を 2 つ以上指定することはできませんので注意してください．

参照　GET@

サンプルプログラム

```
100 DIM A%(153)
110 FOR I=0 TO 153:READ A%(I):NEXT
120 CLS
130 FOR Y=10 TO 490 STEP 20
140 FOR X=10 TO 660 STEP 25
150 PUT@(X,Y),A%
160 NEXT:NEXT:END
170 DATA 22,19,7680,0,0,7680,0,0,0,0,32640,0,0,32640,0,
0,0,0,-64,0,0
180 DATA -64,0,0,0,0,-32,13056,13056,-32,1,0,0,1,-8480,
21120,29568,-32,1,0
190 DATA 0,1,-16,24960,24960,-16,3,0,0,3,-16,0,0,-16,3,
0,0,3,-32656,32640
200 DATA 32640,-16,3,0,0,3,56,-16192,-16192,-16144,7,0,
0,3,-32644,32640
210 DATA 32640,-16,15,0,0,3,-2,0,0,-16,31,0,0,3,-2,1612
8,16128,-16,31,0
220 DATA 0,3,-13,32640,32640,-16,51,0,0,3,-16,-64,-64,-
16,3,0,0,3,-16,-32
230 DATA -32,-16,3,1,1,3,-16,-32,-32,-16,3,1,1,3,-16,-7
712,-7712,-16,3,1,1,3
240 DATA 7680,-7696,-7696,-16,0,3,3,3,3072,-7684,-7684,
-4612,0,15,15,15
```

RANDOMIZE

ステートメント

機 能　新しい乱数系列を設定する.

書 式　RANDOMIZE〔<式>〕

文 例　RANDOMIZE

解 説　新しい乱数の種（seed）を与えることによって，RND 関数で得られる乱数系列を変更します.

<式> には，乱数の種として整数値あるいは実数値を指定します．変数でも定数でもかまいません．もし <式> が省略されると，"Random number seed (−32768 to 32767) ?" というメッセージが出力されて，乱数の種を要求してきます.

RND 関数は，BASIC 起動のたびにまったく同じ乱数系列を発生します.

BASIC 起動のたびに乱数系列を変化させるためには，プログラムのはじめで RANDOMIZE 文を実行させ，毎回違う乱数の種を与えてやらなければなりません．このとき，<式>として TIMER システム変数を使用すると，乱数の種をいちいちキー入力する必要がなくなりますので便利です.

参 照　RND, TIMER

サンプルプログラム

```
100 RANDOMIZE TIMER
110 GOSUB *RNDSUB
120 RANDOMIZE
130 GOSUB *RNDSUB
140 END
150 *RNDSUB'-- rnd number print
160  FOR I=0 TO 9
170   PRINT INT(RND*10);
180  NEXT I:PRINT
190 RETURN
```

```
run
 0  1  8  4  3  4  7  0  9  1
Random number seed (-32768 to 32767)? 123
 6  7  6  3  2  7  4  1  9  1
Ok
```

READ
ステートメント

機　能　DATA文よりデータを読み，変数に割り当てる．

書　式　READ 〈変数〉〔,〈変数〉・・・〕

文　例　READ K, M, S$

解　説　READ文はいつでもDATA文と組み合わせて使わなければなりません．READ文はDATA文のデータを1対1対応の方法で変数に割りあてていきます．〈変数〉はDATA文中のデータを読み込むための変数で，数値変数でも文字変数でもかまいませんが読み出す値と変数の型とは一致していなければなりません．もし，型が一致しない場合には，"Syntax error"が起こります．

ただし，DATA文中のデータが数値である場合は，読み込む変数の型は何でもよく，読み込まれた時点でそのデータはその変数の型となります．

1つのREAD文が1つまたはそれ以上のDATA文を参照したり，またいくつかのREAD文が1つのDATA文を参照したりすることができます．

この目的のためには，READ文で読み込むDATA文を行単位で指定するRESTORE文を使うことができます．

〈変数〉の数がDATA文のデータの個数を越えてしまった場合は，"Out of DATA"のエラーメッセージが表示されます．指定された変数がDATA文のデータの個数よりも少ない場合には，余分のデータは無視されます．

参　照　RESTORE, DATA

サンプルプログラム

```
100 DIM NA$(5),T1$(5),T2$(5),T3$(5)
110 FOR I= 1 TO 5
120 READ NA$(I),T1$(I),T2$(I),T3$(I)
130 NEXT
140 INPUT "push 1-5";J
150 PRINT NA$(J),T1$(J);"-";T2$(J);"-";T3$(J)
160 GOTO 140
170 DATA seiko,03,243,0010
180 DATA akina,0490,233,1111
190 DATA naoko,0471,83,2953
200 DATA yosie,044,944,3151
210 DATA yuu,0474,330,2010
```

```
run
push 1-5? 2
akina           0490-233-1111
push 1-5? 3
naoko           0471-83-2953
push 1-5? 5
yuu             0474-330-2010
push 1-5?
```

REM
ステートメント

機 能 プログラムに注釈を入れる．

書 式

REM '	〔〈注釈文〉〕

文 例 REM TESTPROGRAM

'TESTPROGRAM

解 説 〈注釈文〉には，どんな文字列でも使用することができます．

REM 文は非実行文であり，この文を置いたプログロム行をプログラムの実行に影響を与えないコメント行とすることができます．リストを取ると入力した内容がそのまま出力されます．ただし，REM 文は GOTO 文，GOSUB 文の飛び先として使うことは可能です．

REM 文ではキーワード〝REM〟の代わりにアポストロフィ（'）を使うことができます．

マルチステートメント中で REM 文を使用するときは，REM 文はその行の最後に書かなければなりません．

サンプルプログラム

```
list
100 '  REM ｽﾃｰﾄﾒﾝﾄ ﾆ ﾖｯﾃ
110 ' ﾌﾟﾛｸﾞﾗﾑ ﾉ ﾅｶﾆ
120 ' ｺﾒﾝﾄ ｶﾞ ｶｹﾏｽ.
130 '
140 '  REM ﾊ ｺﾉ ﾚｲ ﾉ ﾖｳﾆ
150 ' "'" ﾃﾞ ﾀﾞｲﾖｳ ﾃﾞｷﾏｽ.
160 END
Ok
run
Ok
```

RENUM

コマンド

機 能 プログラムの行番号を整理する.

書 式 RENUM〔〈新行番号〉〕〔,〈旧行番号〉〕〔,〈増分〉〕

文 例 RENUM

解 説 〈新行番号〉は,新しくつける行番号の最初の行番号で,省略値は10です.

〈旧行番号〉は行番号のつけ替えをはじめる現在のプログラムの行番号です.省略値はそのプログラムの最初の行番号です.

〈増分〉は新しく付ける各行番号のあいだの増分で,省略値は10です.

RENUMコマンドは,また,GOTO,GOSUB,THEN,ON~GOTO,ON~GOSUBおよびERL変数などで参照している行番号も新しい行番号に対応して変更します.これらの文が参照している行番号の行がもし存在しない場合には"Undefined line xxxxx in yyyyy"とエラーメッセージが表示されます.この場合,誤った行番号(xxxxx)はRENUMコマンドによって変更されませんが,行番号yyyyyは変更されます.

注意:RENUMコマンドをプログラム行の順序に変えるのに使うことはできません(たとえば,10,20,30の3つの行がある場合のRENUM 15,30など).また65529以上の行番号を発生することもできません.このような場合には"Illegal function call"エラーが起こります.

サンプルプログラム

```
list
100 ' RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｶﾗ
110 ' ｷﾞｮｳﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴﾏｽ.
120 '
130 PRINT " RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｶﾗ"
140 PRINT " ｷﾞｮｳﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴﾏｽ."
150 END
Ok
renum 200,130,20
Ok
list
100 ' RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｶﾗ
110 ' ｷﾞｮｳﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴﾏｽ.
120 '
200 PRINT " RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｶﾗ"
220 PRINT " ｷﾞｮｳﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴﾏｽ."
240 END
Ok
run
 RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｶﾗ
 ｷﾞｮｳﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴﾏｽ.
Ok
```

RESET
ステートメント

機能	すべてのファイルを閉じる.
書式	RESET
文例	RESET
解説	開かれているすべてのファイルを閉じます．パラメータなしの CLOSE 文と同じ機能です.
参照	CLOSE

サンプルプログラム

```
100 OPEN "ai" FOR OUTPUT AS #1
110 OPEN "scrn:" FOR OUTPUT AS #2
120 PRINT #1,"DISK"
130 PRINT #2,"SCREEN"
140 RESET
run
SCREEN
Ok
```

RESTORE

ステートメント

機能	READ 文で読む DATA 文の行番号を指定する.
書式	RESTORE〔<行番号>〕
文例	RESTORE 800
解説	<行番号> を指定すると，次の READ 文は指定された行の DATA 文からデータを読み始めます. <行番号> が省略されると，次の READ 文はプログラム中の最初の DATA 文から読み始めます.
参照	DATA, READ

サンプルプログラム

```
100  READ A,B,C
110 RESTORE
120  READ D,E,F
130 RESTORE *STRDATA
140  READ A$,B$,C$
150  PRINT A,B,C
160  PRINT D,E,F
170  PRINT A$,B$,C$
180 END
190  DATA 1,2,3
200  DATA 4,5,6
210 *STRDATA
220  DATA aa,bb,cc

run
 1              2              3
 1              2              3
aa             bb             cc
Ok
```

RESUME

ステートメント

機能	エラー処理終了後，プログラムの実行を再開する．

書式

```
RESUME | 〔0〕      |
       | NEXT      |
       | 〈行番号〉 |
```

文例

RESUME 0
RESUME NEXT
RESUME 1000

解説

プログラムの実行を再開する場所に応じて次のように3つの書式を選ぶことができます．

RESUME RESUME 0	エラーの原因となった文からプログラムの実行が再開されます．
RESUME NEXT	エラーの原因となった文のすぐ次の文から実行されます．
RESUME 〈行番号〉	〈行番号〉で指定した行から実行が再開されます．

なお，行番号 xxxx 中に RESUME 0 があるときに RENUM コマンドによって行番号の変更をすると，〝Undefind line 0 in xxxx〞とエラーが出力されますが，これは無視してかまいません．

参照

ON ERROR GOTO

サンプルプログラム

```
100  ON ERROR GOTO *ERRSUB
110 *ENTRY
120  INPUT "X,Y";X,Y
130  A=X/Y
140  PRINT "X/Y =";A
150  GOTO *ENTRY
160 *ERRSUB
170  PRINT "** ｹｲｻﾝ ﾃﾞｷﾏｾﾝ **
180  RESUME *ENTRY
```

```
run
X,Y? 123,56
X/Y = 2.1964285714286
X,Y? 541,0
** ｹｲｻﾝ ﾃﾞｷﾏｾﾝ **
X,Y?
Break in 120
Ok
```

RETURN
ステートメント

機能	サブルーチンからリターンする.
書式	RETURN 〔〈行番号〉〕
文例	RETURN

解説

〈行番号〉が省略されたとき（これが一般的な使い方です）は，そのサブルーチンをコールしたGOSUB文の次の文にリターンします.

〈行番号〉をつけると，指定された行番号に強制的にリターンします. この使用法はおもにON KEY (n) GOSUB文などで指定される割込み処理ルーチンから，特定の行番号にリターンする際に利用されるもので，これによると，割込みの発生箇所に無関係に一定の処理を行うことができます．ただし，例えばFOR～NEXTループ中の割込みによってループの抜け出しが起こり，ループ変数やスタックが破壊されて，プログラムが予期しない結果をもたらすことがありますのでご注意ください.

サンプルプログラム

```
100 MOUSE(2) ON
110 MOUSE(3) ON
120 ON MOUSE(2) GOSUB 160
130 ON MOUSE(3) GOSUB 170
140 A=0:B=+1
150 PRINT A;:A=A+B:GOTO 150
160 BEEP:B=-1:RETURN 150
170 BEEP:B=+1:RETURN 150
```

RIGHT$

関数

機能 文字列の右側から任意の長さの文字列を返す.

書式 RIGHT$ (〈文字列〉, 〈数式〉)

文例 PRINT RIGHT$ (A$, 4)

解説 〈文字列〉の右側から〈数式〉バイト分の長さの文字列を返します.

〈数式〉の値は0から255までの範囲になければなりません.〈数式〉が〈文字列〉の総バイト数以上の場合は,〈文字列〉のすべてを結果とします.

〈数式〉の値が0ならば,ヌルストリングを結果とします.

〈文字列〉に2バイトキャラクタ(日本語文字など)を含んでいる場合,〈数式〉の値によっては2バイトキャラクタを2分してしまい,RIGHT$関数は予期せぬ文字列を返すことがありますので,語境界にはじゅうぶんご注意ください.

参照 LEFT$, MID$ (関数), MID$ (ステートメント)

サンプルプログラム

```
100 ' ﾓｼﾞﾚﾂ ﾉ ｶｲﾃﾝ ( -> )
110  INPUT A$
120  FOR I=0 TO 10
130   PRINT A$
140   B$=RIGHT$(A$,1)
150   A$=B$+MID$(A$,1,LEN(A$)-1)
160  NEXT I
170 END

run
? ABCDEFGHIJKL
ABCDEFGHIJKL
LABCDEFGHIJK
KLABCDEFGHIJ
JKLABCDEFGHI
IJKLABCDEFGH
HIJKLABCDEFG
GHIJKLABCDEF
FGHIJKLABCDE
EFGHIJKLABCD
DEFGHIJKLABC
CDEFGHIJKLAB
Ok
```

RMDIR
コマンド

機 能　ディレクトリを削除する.

書 式　RMDIR〔<ドライブ名>〕<パス名>

文 例　RMDIR "B: TEST. DIR"

解 説　<ドライブ名>で指定されるディスクの, <パス名>で指定されるディレクトリを削除します. <ドライブ名>が省略された場合には, カレントドライブが仮定されます.

指定したディレクトリ内にユーザーの作成したファイルが1つでもあると, "Path/File Access Error" (パス/ファイルアクセスエラー) となり, ディレクトリの削除は行われません. このような場合は, KILLコマンドでそのディレクトリ中のユーザーファイルをすべて削除してからRMDIRコマンドを実行してください.

なお, RMDIRコマンドは, カレントディレクトリ自身を削除することはできません. CHDIRコマンドで他のディレクトリ(ふつうはすぐ上のディレクトリ)に移動してから, RMDIRコマンドを使用するようにしてください.

詳しくは, BASICユーザーズガイドをご覧ください.

サンプルプログラム

```
100 'このプログラムはＭＫＤＩＲの
110 'サンプルプログラムで作成したディレクトリを
120 '削除します
130 CHDIR "¥TEST"
140 PRINT "ＴＥＳＴ２を削除します。"
150 RMDIR"TEST2"
160 PRINT "ＴＥＳＴ３を削除します。"
170 RMDIR"TEST3"
180 CHDIR "¥"
190 PRINT "ＴＥＳＴを削除します。"
200 RMDIR"TEST"

files
A:¥TEST
        .   <DIR>          ..  <DIR> TEST2        <DIR> TEST3        <DIR>
 347136 Bytes free
Ok
run
ＴＥＳＴ２を削除します。
ＴＥＳＴ３を削除します。
ＴＥＳＴを削除します。
Ok
```

RND
関数

機能	乱数を返す.
書式	RND〔(<数式>)〕
文例	R=RND(1)
解説	0以上1未満の乱数を与えます. 発生する乱数は, BASICが起動されるごとに同系列をとります. ただし, RANDOMIZE文によってその系列をかえることもできます.

発生される乱数は〈数式〉の値によって次のように異なります.
〈数式〉が負の場合――新しい乱数系列を発生します.
〈数式〉が0の場合――1つまえに発生した乱数の値を繰り返します.
〈数式〉が正の場合――乱数系列中の次の乱数を発生します.
(〈数式〉)が省略された場合は正の場合と同じで, 次の乱数を発生します.

参照 RANDOMIZE

サンプルプログラム

```
100 DIM SUM(6)
110 FOR I=1 TO 100
120  DA=INT(RND*6+1)
130  SUM(DA)=SUM(DA)+1
140 NEXT I
150 FOR I=1 TO 6
160  PRINT I"-";SUM(I);"%"
170 NEXT I

run
 1 - 13 %
 2 - 29 %
 3 - 16 %
 4 - 19 %
 5 - 13 %
 6 - 10 %
Ok
```

RUN
コマンド

機　能

1）メモリ上にあるプログラムの実行を開始する．

2）ディスクからプログラムファイルをメモリにロードし，そのプログラムを実行する．

書　式

1）RUN〔<行番号>〕

2）RUN <ファイルスペック>〔，R〕

文　例

1）RUN 100

2）RUN "A : TEST"

解　説

1）<行番号>を指定すると，その行から実行がはじまります．指定のない場合には，最も小さい行番号の行から実行がはじまります．プログラムの実行が終るとBASICはいつでもコマンドレベルに戻ります．

2）<ファイルスペック>で指定されるプログラムをディスクからロード後，実行を開始します．

<ファイルスペック>で指定するプログラムファイルは，SAVE文によってディスク上にセーブされた，拡張子に".BAS"をもつファイルでなくてはなりません．しかし，RUN文では<ファイルスペック>中のファイル名に拡張子".BAS"を指定する必要はありません．

<ファイルスペック>中のドライブ名，パス名が省略されたときには，それぞれカレントドライブ，カレントディレクトリが仮定されます．

RUN文を実行すると，すべての開いているファイルは閉じられ，指定されたプログラムがロードされる前にメモリの内容はクリアされます．

Rオプションをつけた場合には，そのとき開かれているファイルはそのままの状態で，実行が開始されます．

サンプルプログラム

```
list
Ok
run "test9"
１００行
１１０行
１３０行
Ok
```

```
list
100 PRINT "１００行"
110 PRINT "１１０行"
120 *ENTRY
130 PRINT "１３０行"
Ok

run
１００行
１１０行
１３０行
Ok
run 110
１１０行
１３０行
Ok
run *entry
１３０行
Ok
```

SAVE
コマンド

機 能 メモリ上のプログラムをディスクにセーブする.

書 式 SAVE <ファイルスペック>〔, |A| 〕
|P|

文 例 SAVE "B : ¥BIN¥ MYPROG", A

解 説 メモリ上にあるプログラムを,<ファイルスペック>で指定されたファイル名でディスクに格納します.

<ファイルスペック>中のドライブ名,パス名が省略されたときには,それぞれカレントドライブ,カレントディレクトリが仮定されます.

指定したファイル名と同じ名前のファイルが同じディレクトリ内に存在した場合には,古いファイルは失われて,新しいものに変更されます.

A オプションが指定された場合には,プログラムはアスキー形式でセーブされます.A オプションの指定がない場合には,バイナリ形式でセーブされます.

アスキー形式でセーブしたファイルは,バイナリ形式でセーブしたファイルに比べてサイズ(ディスク上で占めるバイト数)が大きくなります.しかし,ディスク上のプログラムファイルをデータとして LINE INPUT #文で読み込んだり,MERGE コマンドでメモリ上のプログラムとマージ(混合)したりするためには,そのプログラムファイルはアスキー形式でセーブされたものでなくてはなりません.

P オプションが指定された場合には,プログラムは暗号化されたバイナリ形式でセーブされます.P オプションは,セーブされたプログラムを,読出し,書込み,変更といった操作から保護するための機能をもっています.したがって P オプションつきでセーブされたプログラムをメモリにロードしても,LIST や EDIT によって,内容を見たり,変更しようとしても,"Illegal function call" エラーとなります.

一度,P オプションにより保護が指定されると,これを解除する方法は用意されていませんので,二度と内容の変更などはできなくなります.ただし,KILL 文によりそのプログラムファイルを削除することは可能です.

参 照 LOAD, MERGE, SAVE

サンプルプログラム

```
save "sample1"
save "a:sample2",a
save "d:sample3",p
```

SCREEN

ステートメント

機 能　画面に対して種々のモードを設定する.

書 式　SCREEN〔〈カラースイッチ〉〕〔,〈縦横スイッチ〉〕〔,〈キャラクタモードスイッチ〉〕〔,〈カナ漢字変換スイッチ〉〕

文 例　SCREEN 1, 0, 0, 1

解 説　カラーモードにするか白黒モードにするか，画面を縦表示にするか横表示にするか，など，ディスプレイ画面に対する種々のモードを設定します.

〈カラースイッチ〉は，次のように指定します.

0：白黒モード

1：カラーモード

このスイッチを0（白黒モード）にセットした状態でPALETTE文を使用すると〝Illegal function call〞エラーが生じます．また，このときCOLOR文などの各グラフィックコマンドで使用する〈パレット番号〉に，0を指定すれば黒，0以外を指定すれば白がそれぞれ指定されたものとみなされます.

なお，カラーボードを実装していないときに〈カラースイッチ〉を1（カラーモード）にすることはできません．〝Illegal function call〞エラーとなります.

〈縦横スイッチ〉は，次のように指定します.

0：横表示

1：縦表示

ディスプレイ画面の縦横表示状態は，ディスプレイ本体を横置きあるいは縦置きにすることにより，ディスプレイ本体の内部スイッチによって自動的に横表示，縦表示に切り換わるようになっていますが，この〈縦横スイッチ〉によってBASICからソフト的にも切り換えを行うことができます.

〈キャラクタモードスイッチ〉は，次のように指定します.

0：漢字キャラクタモード

1：グラフィックキャラクタモード

漢字キャラクタモードでは，内部連続コードが&H81～&H9Fおよび&HE0～&HEFの範囲を2バイトキャラクタ（漢字など）のシフトJISコードの第1バイトとして扱います.

グラフィックキャラクタモードでは，内部連続コードが&H80～&H9Fおよび&HE0～&HFFの範囲をグラフィックキャラクタとして扱います.

したがって，漢字として入力された文字列や，グラフィックキャラクタとして入力された文字列を表示する場合，この〈グラフィックスイッチ〉により，適合したモードに切り換えないとそれらの

文字列を正しく表示することができません。

なお，WIDTH 文，あるいは WIDTH USING 文によって設定された現在のキャラクタフォントが 8 ×16の大きさであるとき以外は，漢字表示をすることができませんのでご注意ください。

〈カナ漢字変換スイッチ〉は，次のように指定します。

　0：カナ漢字変換しない

　1：カナ漢字変換する

1を指定すると，ディスプレイ画面の下部2行分が空けられ，キーボード上の[漢字]キーを押すことにより漢字入力用テンプレートがこの箇所に表示されます。漢字入力のしかたについては BASIC ユーザーズガイドまたは MS-DOS ユーザーズガイドをご覧ください。〈キャラクタモードスイッチ〉が1でグラフィックキャラクタモードになっているときに，〈カナ漢字変換スイッチ〉に1を指定すると，画面の下部2行分が空けられますが，漢字入力用テンプレートは表示できません。

また，WIDTH USING 文を使用して画面の設定を行ったときは，漢字入力用テンプレートの表示をすることはできません。

SCREEN 文では，それぞれのパラメータを省略することができ，省略時には以前に指定された値をとります。しかし，すべてのパラメータを省略することはできませんのでご注意ください。

また，SCREEN 文により，モードの切り換えが行われたときは，LP はビューポートの中央の点に移動します。

参照　COLOR, PALETTE, WIDTH, WIDTH USING

サンプルプログラム

```
100 LOCATE 1,1,0
110 SCREEN 0:PRINT "白黒画面":GOSUB DISPLAY
120 SCREEN 1:PRINT "カラー画面":GOSUB DISPLAY
130 SCREEN 0,1:PRINT "水平画面":GOSUB DISPLAY
140 SCREEN 0,0,0:PRINT "垂直画面":GOSUB DISPLAY
150 CLS
160 SCREEN 0,0,1:PRINT "ｸﾞﾗﾌｨｯｸｷｬﾗｸﾀﾓｰﾄﾞ":GOSUB DISPLAY
170 SCREEN 1,0,0,1
180 END
190 DISPLAY
200 LINE (100,100)-STEP(200,200),13,BF
210 LOCATE 7,7,0:PRINT"“画面モードの設定”"
220 FOR I=1 TO 4000:NEXT
230 BEEP
240 RETURN
```

SCREEN
関数

機能　指定されたキャラクタ座標上のキャラクタコードを返す.

書式　SCREEN (Y, X)

文例　C=SCREEN (10, 15)

解説　ディスプレイ画面のキャラクタ座標 (Y, X) 上に表示されているキャラクタの内部連続コードを返します.

Y は垂直座標を表し, X は水平座標を表します.

表示されているキャラクタは, 1バイトキャラクタでも2バイトキャラクタでもかまいません. 2バイトキャラクタの場合, 1バイト目（左半分）のキャラクタ座標でも2バイト目（右半分）のキャラクタ座標でもそのキャラクタの内部連続コードが返されます.

画面上の空白部分には, すべてスペースコードが埋まっているものと解釈されますので, これらの座標を参照すると "32"（スペースコード）が返されます.

参照　LOCATE

サンプルプログラム

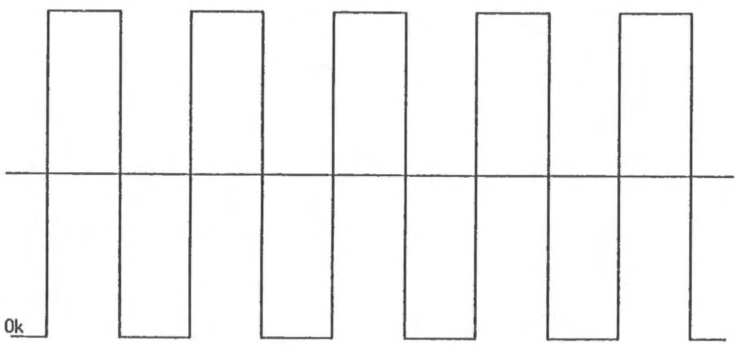
```
100 DEFINT A-Z:WIDTH 80,25:LOCATE ,,0
110 SCREEN 1,0,0,0
120 COLOR 2
130 FOR I=1 TO 25
140   FOR J=1 TO 80
150     PRINT CHR$(RND*25+65);
160   NEXT
170 NEXT
180 Y=RND*24+1:X=RND*79+1
190 A=SCREEN(Y,X)
200 IF A=67 OR A=69 OR A=78 THEN 210 ELSE 220
210 LOCATE Y,X:COLOR 13:PRINT CHR$(A);:COLOR 2:GOTO 180
220 LOCATE Y,X :PRINT " ";:GOTO 180
```

SGN
関数

機　能	数式の正負を調べる．
書　式	SGN（〈数式〉）
文　例	PRINT SGN (A)
解　説	〈数式〉が正の場合は１を，〈数式〉が０の場合は０を，〈数式〉が負の場合には，－１を返します．

サンプルプログラム

```
100  SCREEN 1:CLS:LOCATE 20,1,0
110  LINE (0,250)-(639,250),9
120  X=5
130  FOR I=0 TO 31.42 STEP .05
140   S=SGN(INT(COS(I)*20))
150   ON S+2 GOSUB *PLUS,*ZERO,*MINUS
160   X=X+1
170  NEXT I
180 END
190 *PLUS
200  PSET (X,100),13:RETURN
210 *ZERO
220  LINE (X,100)-(X,400),13:RETURN
230 *MINUS
240  PSET (X,400),13:RETURN
```



SHELL
コマンド

機能	BASIC の状態を離れることなく DOS コマンドを実行する，あるいは DOS に制御を移す．
書式	SHELL〔"<文字列>"〕
文例	SHELL "B : DIR"

解説

SHELL コマンドは BASIC の状態を壊すことなく，DOS のコマンドを実行したり，DOS に制御を移したりするときに使用します．

<文字列> には DOS のコマンド列を指定します．これは DOS レベルで入力可能なコマンドであれば，内部コマンドでも外部コマンドでもかまいません．BASIC は現在のテキストエリアにあるプログラムや変数の値をそのままの状態に保ったまま，<文字列> で指定された DOS のコマンドを実行します．DOS のコマンドの実行が終了すれば制御は BASIC にもどります．

なお，この場合，DOS のコマンドによって画面に出力されるキャラクタは，すべてグラフィックスとして出力されるため，BASIC のスクリーンエディタでこれらの文字に編集作業を加えることはできません．

<文字列>を省略して SHELL コマンドを実行すると，BASIC の現在の状態が保たれたまま DOS に制御が移り，DOS のコマンドレベルとなります (DOS のプロンプトが出る)．この状態では，ユーザーは BASIC の存在を意識することなく，DOS 上での操作ならば何でも行うことができます．この状態からもとの BASIC のレベルにもどるには，DOS 上で"exit"コマンドを入力します．DOS は再び制御を BASIC にもどします．

SHELL コマンドの実行後に BASIC に制御がもどってきた場合，BASIC のプログラムや変数の値は SHELL コマンド実行前の状態のままとなっていますから，再びそのプログラムを走らせたり，変数の値を使用したりすることができます．

SHELL コマンドは，DOS のコマンドプロセッサを，BASIC のチャイルドとして走らせるものです．つまり，SHELL コマンドはメモリ上にロードされている BASIC およびユーザーの BASIC プログラムのエリア外に，DOS のコマンドプロセッサをロードして制御を渡すので，BASIC そのものの状態はまったく壊されることがないのです．ただし，広いメモリ空間を必要としますので，"Out of memory" エラーとなることがあります．

この SHELL の概念やチャイルドプロセスなどの詳細については，BASIC ユーザーズガイド，および MS-DOS ユーザーズガイドをご覧ください．

なお，ファイルを開いたまま，SHELL コマンドを使って DOS レベルのコマンドでそのファイルに操作を加えたりすると，ファイルが破壊されることがあります．SHELL コマンドの実行前には必ずファイルを閉じるようにしてください．

SHELL

注意：<文字列>に“BASIC. EXE”を指定することはできません．つまり，BASICそのものをチャイルドとして走らせることはできません．

参照　SYSTEM

サンプルプログラム

```
shell "dir b:"
Ok
 ドライブ B: のディスクにはボリューム ラベルがありません
 ディレクトリは  B:¥

TEST1    BAS       11  83-08-29  18:25
TEST2              22  83-08-29  18:27
TEST3    BAS       11  83-08-29  18:25
TEST4    BAS      434  83-09-02  17:08
TEST5    BAS       11  83-08-29  18:25
TEST7              26  83-08-31  15:14
STEST    DAT       75  83-08-31  14:52
TEST8              33  83-08-31  16:00
TEST10            105  83-08-31  18:39
TEST11            453  83-08-31  20:23
        10 個のファイルがあります
    352256 バイトが使用可能です
```

SIN
関数

機能	正弦値（サイン）を返す.
書式	SIN（<数式>）
文例	PRINT SIN (1)
解説	<数式>の値に対する正弦値を返します.〈数式〉の単位はラジアンです. また,〈数式〉にはどんな型の数値でも指定できますが, SIN 関数は常に倍精度の値を返します.
参照	COS, TAN

サンプルプログラム

```
100 ' csec ヲ モトメル
110  PI=1.5707963267949#
120  F$="csc(###° )=###.#####
130  INPUT "カクド  :";D
140  R=D MOD 180
150   IF R=0 THEN PRINT "テイギサレマセン":END
160  C=1/SIN(R/180*PI)
170  PRINT USING F$;D,C
180 END
```

```
run
カクド  :? 45
csc( 45° )=  2.61313
Ok
```

SOUND

ステートメント

機能　内蔵スピーカから音を出す．

書式　SOUND 〈周波数〉，〈長さ〉

文例　SOUND 880，30

解説　〈周波数〉で指定される高さの音を〈長さ〉で指定される長さだけ，内蔵スピーカから出力します．

〈周波数〉の単位は Hz（ヘルツ）で，220～7040の範囲で指定します．

〈長さ〉は1単位約55ms で，0～65535の範囲で指定します．ただし，3000が最長ですから，3001以上を指定してもそれ以上長くはなりません．

PLAY 文と異なり，周波数の刻みが平均律（2の12乗根を等比とする等比数列）ではありませんから，自由な高さの音を出すことができます．

参照　PLAY

サンプルプログラム

```
100 SS=220
110 FOR S=1 TO 17
120   SOUND SS,1
130   SS= SS ^1.03
140 NEXT
```

SPACE$
関数

機能	任意の長さのスペース文字列を返す.
書式	SPACE$ (<数式>)
文例	PRINT SPACE$ (10) ; "ABC"
解説	<数式>の数だけのスペース (CHR$(32)) をもった文字列を返します. <数式>の値は 0 から255までの範囲になければなりません.
参照	SPC

サンプルプログラム

```
100 '2ｼﾞｷｮｸｾﾝ
110  FOR I=-6 TO 6
120   A$="|"+SPACE$(ABS(I*I))+"*"
130   PRINT A$
140  NEXT I
150 END
```

```
run
|                                    *
|                         *
|                *
|         *
|    *
| *
|*
| *
|    *
|         *
|                *
|                         *
|                                    *
Ok
```

SPC
関数

機能 任意の数だけスペースを出力する.

書式 SPC (<数式>)

文例 PRINT SPC (10) ; A$

解説 <数式>の数だけスペースを出力します. SPC 関数は PRINT 文や LPRINT 文などの出力文中でのみ使用することができます.

<数式>の値は, −32768から32767まで許されますが, 負の数はすべて 0 とみなされます. また正の数の場合でも, 現在の水平表示文字数以上の場合は, <数式>をその水平表示文字数で割った余りを値とします. つまり, 現在の水平表示文字数が80であるとすれば, 次の 2 つの例は同じ結果を返します.

```
PRINT "ABC" ; SPC(30) ; "DEF"
PRINT "ABC" ; SPC(110) ; "DEF"
```

ただし, 次の文字列が表示されるべき位置にスペース以外の文字が存在しているときには, 次の文字列は次の行に表示されます. 例えば, 現在の水平表示数が80のときに,

```
PRINT "A" ; SPC(79) ; "XYZ"
```

を実行すると, "XYZ" は "A" の表示される行の次の行に出力されます.

参照 SPACE$, TAB

サンプルプログラム

```
100  FOR I=0 TO 10
110   PRINT "PC-100";SPC(I);"N100-BASIC"
120  NEXT
130 END

run
PC-100N100-BASIC
PC-100 N100-BASIC
PC-100  N100-BASIC
PC-100   N100-BASIC
PC-100    N100-BASIC
PC-100     N100-BASIC
PC-100      N100-BASIC
PC-100       N100-BASIC
PC-100        N100-BASIC
PC-100         N100-BASIC
PC-100          N100-BASIC
Ok
```

SQR
関数

機　能　平方根を返す．

書　式　SQR（<数式>）

文　例　A＝SQR(2)

解　説　<数式>で指定された値の平方根を返します．

<数式>は正の値でなくてはなりません．負の値を指定すると〝Illegal function call〞エラーとなります．

<数式>には，どんな型の数値を指定しても，返される値は倍精度実数となります．

サンプルプログラム

```
100 'ﾙｰﾄ ﾉ ﾋｮｳ
110 PRINT " << SQUARE ROOT TABLE >>
120 FOR N=0 TO 20
130   PRINT "N=";N,"ROOT(N)=";
140   PRINT USING "##.#####";SQR(N)
150 NEXT N
```

```
run
 << SQUARE ROOT TABLE >>
N= 0          ROOT(N)= 0.00000
N= 1          ROOT(N)= 1.00000
N= 2          ROOT(N)= 1.41421
N= 3          ROOT(N)= 1.73205
N= 4          ROOT(N)= 2.00000
N= 5          ROOT(N)= 2.23607
N= 6          ROOT(N)= 2.44949
N= 7          ROOT(N)= 2.64575
N= 8          ROOT(N)= 2.82843
N= 9          ROOT(N)= 3.00000
N= 10         ROOT(N)= 3.16228
N= 11         ROOT(N)= 3.31662
N= 12         ROOT(N)= 3.46410
N= 13         ROOT(N)= 3.60555
N= 14         ROOT(N)= 3.74166
N= 15         ROOT(N)= 3.87298
N= 16         ROOT(N)= 4.00000
N= 17         ROOT(N)= 4.12311
N= 18         ROOT(N)= 4.24264
N= 19         ROOT(N)= 4.35890
N= 20         ROOT(N)= 4.47214
Ok
```

STOP
ステートメント

機能　プログラムの実行を一時停止してコマンドレベルに戻る.

書式　STOP

文例　STOP

解説　STOP 文はプログラムの実行を一時停止するためのもので，プログラムのどこで使用してもかまいません．STOP 文を実行すると，次のメッセージが表示されます．

BREAK in xxxx　　(xxxx はストップした行番号)

STOP 文が実行されると，BASIC は常にコマンドレベルに戻りますが，それまでに実行されたプログラムの状態，例えば，変数の値などはすべて保持されたままとなります．

したがって，STOP 文でプログラムの実行を一時停止させてダイレクトモードで変数の値を調べ，CONT コマンドで実行を再開し，次の STOP 文でまた変数の値を調べる，というような使い方をすることにより，プログラムのデバッグに利用することができます．

ただし，ファイルが開かれているときにも STOP 文はファイルを閉じませんから，注意してください．END 文はファイルを閉じます．

参照　CONT

サンプルプログラム

```
list
100 FOR I=10 TO 19
110  PRINT I,SQR(I)
120  STOP
130 NEXT I
140 END
Ok
run
 10            3.1622776601684
Break in 120
Ok
print sqr(i)*sqr(i)
 10
Ok
cont
 11            3.3166247903554
Break in 120
Ok
```

STR$
関数

機能

数値を表す文字列を返す.

書式

STR$ (<数式>)

文例

A$=STR$ (123)

解説

<数式>によって指定された値を数値表記の文字列に変換します.<数式>は整数型と実数型(倍精度,単精度) のどちらでもかまいません.

注意:STRING$関数と区別すること.

参照

VAL, STRING$

サンプルプログラム

```
100 ' 00:00:00 カラ シテイシタ ビョウスウ ダケ
110 ' ケイカシタ ジコク ヲ モシ゛レツ デ゛ モトメル
120  DEF FNST$(A$)=RIGHT$("0"+RIGHT$(A$,LEN(A$)-1),2)
130  INPUT "秒数:",SS
140  H=SS¥3600:M=(SS-H*3600)¥60
150  S=SS MOD 60
160  H$=STR$(H):M$=STR$(M):S$=STR$(S)
170  H$=FNST$(H$):M$=FNST$(M$):S$=FNST$(S$)
180  PRINT "00:00:00 から ";SS;"秒経過した";
190  PRINT "時刻は ";H$;":";M$;":";S$;" です"
200 END
run
秒数:6874
00:00:00 から  6874 秒経過した時刻は 01:54:34 です
Ok
```

STRING$
関数

機能 任意の文字を任意の数だけ返す.

書式 STRING$（〈数式〉, | 〈文字列〉 / 〈キャラクタコード〉 | ）

文例 A$=STRING$（10, "+"）

解説 〈文字列〉または〈キャラクタコード〉で表されるキャラクタを〈数式〉の数だけ連結した文字列を返します.

〈文字列〉は文字変数, 文字定数どちらでもよいですが, 複数の文字を指定した場合は, 最初の１文字だけが有効となります.

〈キャラクタコード〉は内部連続コードを指定します. １バイトキャラクタでも２バイトキャラクタ（漢字など）でも,〈数式〉の文字数だけそのキャラクタを連結した文字列を返します.

参照 CHR$, STR$

サンプルプログラム

```
100 INPUT A$
110 L=LEN(A$)
120 PRINT STRING$(L+4,"+")
130 PRINT "+ ";SPC(LEN(A$)+1);"+"
140 PRINT "+ ";A$;" +"
150 PRINT "+ ";SPC(LEN(A$)+1);"+"
160 PRINT STRING$(L+4,"+")
170 END
```

```
run
? I love you
++++++++++++++
+            +
+ I love you +
+            +
++++++++++++++
Ok
```

SWAP
ステートメント

機能　2つの変数の値を交換する.

書式　SWAP <変数>, <変数>

文例　SWAP A$, B$

解説　SWAP文は, 2つの <変数> の値を互いに交換します.

2つの <変数> は, どんな型の変数でもかまいませんが, その2つの型が一致していないと "Type mismatch" エラーが起こります.

サンプルプログラム

```
100 ' 2ツ ノ ハイレツ ノ ナイヨウ ヲ イレカエル
110  DIM A(10),B(10)
120  FOR I=1 TO 10
130   A(I)=I*2:B(I)=I*3
140  NEXT I:GOSUB *PRINTTAB
150  FOR I=1 TO 10
160   SWAP A(I),B(I)
170  NEXT I:GOSUB *PRINTTAB
180 END
190 *PRINTTAB
200       PRINT :PRINT:PRINT "A";
210  FOR I=1 TO 10
220   PRINT USING "###";A(I);
230  NEXT
240       PRINT:PRINT"B";
250  FOR I=1 TO 10
260   PRINT USING "###";B(I);
270  NEXT
280 RETURN

run


A  2  4  6  8 10 12 14 16 18 20
B  3  6  9 12 15 18 21 24 27 30

A  3  6  9 12 15 18 21 24 27 30
B  2  4  6  8 10 12 14 16 18 20
Ok
```

SYSTEM
コマンド

機能	DOS にもどる.
書式	SYSTEM
文例	SYSTEM
解説	SYSTEM コマンドは，BASIC を抜け出して，制御を DOS にもどすときに使います．

SYSTEM コマンドが実行されると，開かれているファイルはすべて閉じられてから DOS にもどります．この場合，メモリ中 (BASIC のテキストエリア中) にあるプログラムや変数はすべてクリアされてしまいますのでご注意ください．

BASIC の状態を壊すことなく DOS に制御を移したいときには，SHELL コマンドを用いることができます．

参照　SHELL

サンプルプログラム

```
system
A>basic ¥bas
1
4
9
16
25
36
49
64
81
100
```

```
100 '-- bas program --
110 FOR I=1 TO 10
120 PRINT I^2
130 NEXT
140 SYSTEM
```

TAB

関数

機 能 行頭から任意の位置までスペースを出力する.

書 式 TAB (〈数式〉)

文 例 PRINT TAB (10) ; 〝BASIC〟

解 説 〈数式〉の数だけ，行頭からスペースを出力します．TAB 関数は PRINT 文や LPRINT 文などの出力文中のみで使用されます．

〈数式〉の値は，−32768から32767（0 が行の左端）まで許されますが，負の数はすべて 0 とみなされます．また正の数の場合でも，現在の水平表示文字数以上の場合は，〈数値〉をその水平表示文字数で割った余りを値とします．つまり，現在の水平表示文字数が80であるとすれば，次の 2 つの例は同じ結果を返します．

PRINT 〝ABC〟; TAB (30) ; 〝DEF〟
PRINT 〝ABC〟; TAB (110) ; 〝DEF〟

ただし，次の文字列が表示されるべき位置にスペース以外の文字が存在しているときには，次の文字列は次の行に表示されます．例えば，現在の水平表示文字数が80のときに，

PRINT 〝A〟; TAB (81) ; 〝XYZ〟

を実行すると，〝XYZ〟 は 〝A〟 の表示される行の次の行に出力されます．

SPC 関数と異なり，TAB 関数は常に行頭からの位置を指定できますので，書式をそろえたりするときに便利です．

参 照 SPC

サンプルプログラム

```
100 FOR DEG=0 TO 180 STEP 20
110   RAD=DEG/180*3.141592653589#
120   PRINT "DEG=";DEG;
130   PRINT TAB(15)"SIN(DEG)=";SIN(RAD);
140   PRINT TAB(45)"COS(DEG)=";COS(RAD)
150 NEXT DEG
160 END
run
DEG= 0         SIN(DEG)= 0                 COS(DEG)= .99999999999997
DEG= 20        SIN(DEG)= .34202014332558   COS(DEG)= .93969262078594
DEG= 40        SIN(DEG)= .6427876096864    COS(DEG)= .7660444431191
DEG= 60        SIN(DEG)= .8660254037843    COS(DEG)= .50000000000026
DEG= 80        SIN(DEG)= .98480775301213   COS(DEG)= .17364817766729
DEG= 100       SIN(DEG)= .98480775301227   COS(DEG)=-.1736481776665
DEG= 120       SIN(DEG)= .86602540378469   COS(DEG)=-.49999999999957
DEG= 140       SIN(DEG)= .64278760968697   COS(DEG)=-.76604444311861
DEG= 160       SIN(DEG)= .34202014332635   COS(DEG)=-.93969262078567
DEG= 180       SIN(DEG)= 7.932385E-13      COS(DEG)=-1
Ok
```

TAN
関数

機 能 正接値（タンジェント）を返す．

書 式 TAN（<数式>）

文 例 A＝TAN（123）

解 説 <数式> の値に対する正接値を返します．<数式> の単位はラジアンです．

<数式> には，どんな型の数値を指定しても，返される値は倍精度実数となります．

これとは逆に，正接値から角度（逆正接値）を求めたいときには ATN 関数を用います．

参 照 SIN, COS, ATN

サンプルプログラム

```
100 ' cot ｦ ﾓﾄﾒﾙ
110  PI=3.1415926565898#
120  F$="cot(###°)=###.#####
130  INPUT "ｶｸﾄﾞ :";D
140  R=D MOD 180
150   IF R=0 THEN PRINT "ﾃｲｷﾞｻﾚﾏｾﾝ":END
160  C=1/TAN(R/180*PI)
170  PRINT USING F$;D,C
180 END

run
ｶｸﾄﾞ :? 30
cot( 30°)=  1.73205
Ok
```

TIMES$
システム変数

機 能	内蔵クロックの時刻を返す.
書 式	1）TIME$ 2）TIME$＝ "hh : mm : ss"
文 例	PRINT TIME$

解 説

TIME$はシステム変数であり現在の時刻を文字列として持っています．TIME$の形式は，

HH : MM : SS

で示されます．ここで，HH は時，MM は分，SS は秒を表します．

また，TIME$は次のように新たに時刻を設定することもできます．HH は00から23まで，MM は00から59まで，SS は00から59までの数です．

TIME$＝ "12 : 30 : 00"

参 照 DATE$

サンプルプログラム

```
100 ' シテイ ジカン ゴ ノ ジコク ヲ モトメル
110 T$=TIME$
120 PRINT "現在の時刻は ";T$;"です"
130 INPUT "時間  (0-23):",H
140 INPUT " 分   (0-59):",M
150 INPUT " 秒   (0-59):",S
160 HH=VAL(LEFT$(T$,2))
170 MM=VAL(MID$(T$,4,2))
180 SS=VAL(RIGHT$(T$,2))
190 SS=SS+S:MM=MM+M:HH=HH+H
200 FOR I=0 TO 1
210  MM=MM+SS¥60:SS=SS MOD 60
220  HH=HH+MM¥60:MM=MM MOD 60
230  DD=DD+HH¥24:HH=HH MOD 24
240 NEXT
250 PRINT "求める時刻は"
260  IF DD=1 THEN PRINT "翌日の ";
270 PRINT USING " ##時 ##分 ##秒です";HH,MM,SS
280 END
run
現在の時刻は 16:43:07です
時間 (0-23):20
 分  (0-59):24
 秒  (0-59):53
求める時刻は
翌日の  13時  8分  0秒です
Ok
```

TIMER
システム変数

機　能	午前0時よりの積算秒数を返す.
書　式	TIMER
文　例	A＝TIMER

解　説

TIMER はシステム変数であり，午前0時より現在の時刻に至るまでの積算秒数を実数として持っています.

TIMER は，プログラムのはじめと終わりで参照してそのプログラムの実行時間を計ったり，RANDOMIZE 文の乱数の種として与えたりするのに使います.

TIMER には定数を代入することはできませんが，TIME$と連動していることを利用して，例えば

TIME$＝"00：01：00"

を実行すれば，その瞬間に TIMER は60となります.

参　照　TIME$

サンプルプログラム

```
100 CLS
110 LOCATE 1,1:PRINT "ＴＹＰＥの練習　　大文字で入力してください。"
120 PRINT "ヨーイ":FOR I=1 TO 7000:NEXT:BEEP
130 T=TIMER
140 FOR I=1 TO 10
150 A$=CHR$(RND*25+65)
160 SOUND 1000,2:LOCATE 12,20:PRINT CDBL$(A$)
170 IF INPUT$(1)<>A$ THEN LOCATE 13,20:PRINT "違う":SOUND
2000,10:GOTO 170
180 LOCATE 13,20:PRINT "    "
190 NEXT I
200 TI=TIMER-T
210 IF TI < 5 THEN PRINT "す速いっ！！":END
220 IF TI <10 THEN PRINT "まあまあ速い！" :END
230 IF TI <15 THEN PRINT "人並ですな。":END
240 IF TI <20 THEN PRINT "もっと練習しなさい。":END
250 IF TI <25 THEN PRINT "キー配列を覚えなさいっ。":END
260 PRINT "ＰＣ－１００とのおつきあいは，キーボードから．．．"
270 END
```

```
ＴＹＰＥの練習　　大文字で入力してください。
ヨーイ



                    Ｙ

人並ですな。
Ok
```

TIMER ON/OFF/STOP

ステートメント

機能	タイマによる割込みを許可，保留する．
書式	TIMER ON TIMER OFF TIMER STOP
文例	TIMER ON

解説

ON TIMER(n) GOSUB 文によって設定された時間間隔ごとにかかる割込みを許可（ON）するか，保留（OFF，STOP）するかを設定します．

TIMER ON 文は，タイマ割込みを許可します．このステートメント実行後は，ON TIMER(n) GOSUB 文によって設定された時間間隔ごとに割込みを発生し，同文で定義されている処理ルーチンに分岐します．

TIMER OFF 文，TIMER STOP 文は，タイマ割込みを保留します．このステートメント実行後は，タイマによる割込みがあってもそのことを覚えているだけで処理ルーチンへの分岐は起こりません．しかし，その後 TIMER ON 文によって割込みが許可されると，先程のタイマ割込みによって処理ルーチンに分岐します．

なお，割込み処理ルーチンに制御が移ると自動的に TIMER STOP（割込み保留）状態となりますので，割込み処理ルーチンの先頭で TIMER OFF 文あるいは TIMER STOP 文を実行する必要はありません．

参照　ON TIMER(n) GOSUB

サンプルプログラム

```
100 ON TIMER(5) GOSUB 180
110 TIMER ON:TI=TIMER
120 LOCATE 2,2:PRINT TIMER-TI
130 B$=INKEY$
140 IF B$="s" THEN TIMER STOP:LOCATE 3,2:PRINT"stop"
160 IF B$="o" THEN TIMER OFF :LOCATE 3,2:PRINT"off ":GOSUB 2
00:TIMER ON:LOCATE 3,2:PRINT "on      "
170 GOTO 120
180 BEEP
190 RETURN
200 LOCATE 2,2:PRINT TIMER-TI:A$=INKEY$:IF A$="" THEN 200
210 RETURN
```

TRON/TROFF

コマンド

機能 プログラム実行状態のトレース（追跡）の設定／解除を行う．

書式

```
TRON
TROFF
```

文例

```
TRON
TROFF
```

解説 プログラムのデバッグを行うときに使用します．

TRON コマンドを実行するとプログラムトレースモードとなり，この状態でプログラムを実行すると，実行されたプログラム行の行番号がディスプレイ画面に順次表示されていきます．行番号は，それぞれ角カッコで囲まれて出力されます．

TROFF コマンドを実行するとプログラムトレースモードは解除され，通常の状態にもどります．NEW コマンドでもプログラムトレースモードの解除を行うことができます．

なお，TRON，TROFF 両コマンドはダイレクトモードでも，プログラム中に入れてプログラムモードでも使用することができます．

サンプルプログラム

```
100 'goto
110 GOTO *START
120 PRINT "PC-100 ";:GOTO 160
130 PRINT "COMPUTER ";:GOTO 120
140 *FINE:END
150 PRINT "PERSONAL ";:GOTO 130
160 PRINT "BASIC ";:GOTO *FINE
170 *START
180 PRINT "NEC ";:GOTO 150

Ok
TRON
Ok
run
[100][110][170][180]NEC [150]PERSONAL [130]COMPUTER [120]PC-100 [160]BASIC [140]
Ok
TROFF
Ok
run
NEC PERSONAL COMPUTER PC-100 BASIC
Ok
```

USR
関数

機　能　機械語で作られたユーザー関数を呼び出す．

書　式　USR〔<番号>〕(<引数>)

文　例　I=USR3 (J)

解　説　ユーザーの作成した機械語関数ルーチン（ユーザー関数）を呼び出します．ユーザー関数は，あらかじめメモリ中に準備されており，また DEF USR 文により，その実行開始アドレスが設定されていなければなりません．

<行番号> は 0 から 9 までの値で，DEF USR 文により定義された番号に対応します．<番号> が省略された場合には， 0 と解釈されます．

<引数>により BASIC から機械語ルーチンへの値の受渡しをすることができます．<引数>の受渡し方法については，BASIC ユーザーズガイドを参照してください．

USR 関数により機械語ルーチンを呼び出すときは，前もって DEF SEG 文により，DEF USR 文で USR 関数を定義した時と同じ位置に，セグメントベースを位置付けておかなければなりません．

なお，機械語ルーチンを呼び出す方法としては，USR 関数の他に CALL 文が用意されています．CALL 文では複数の引数を機械語ルーチンに受け渡すことができます．

参　照　DEF USR，CALL

VAL
関数

機能	文字列の表す数値を返す．
書式	VAL（<文字列>）
文例	A=VAL（"2"）
解説	数値表記の<文字列>を数値に変換して返します．もし，<文字列>の最初の文字が+，−，&，または数字でなければ，VALの値は0になります． ただし，文字列中のスペースは無視します．
参照	STR$

サンプルプログラム

```
100 ' 00:00:00 カラノ ケ゛ンサ゛イノ
110 ' シ゛コク マテ゛ノ ヒ゛ョウスウ ヲ モトメル
120  T$=TIME$
130  H=VAL(LEFT$(T$,2))
140  M=VAL(MID$(T$,4,2))
150  S=VAL(RIGHT$(T$,2))
160  SS=H*3600+M*60+S
170  PRINT "00:00:00 から ";T$;" までは";
180  PRINT SS;"秒間です"
190 END

run
00:00:00 から 16:47:07 までは 60427 秒間です
Ok
```

VARPTR
関数

機　能　変数あるいはファイルバッファの先頭アドレスをオフセット値で返す．

書　式　1）VARPTR（<変数名>）

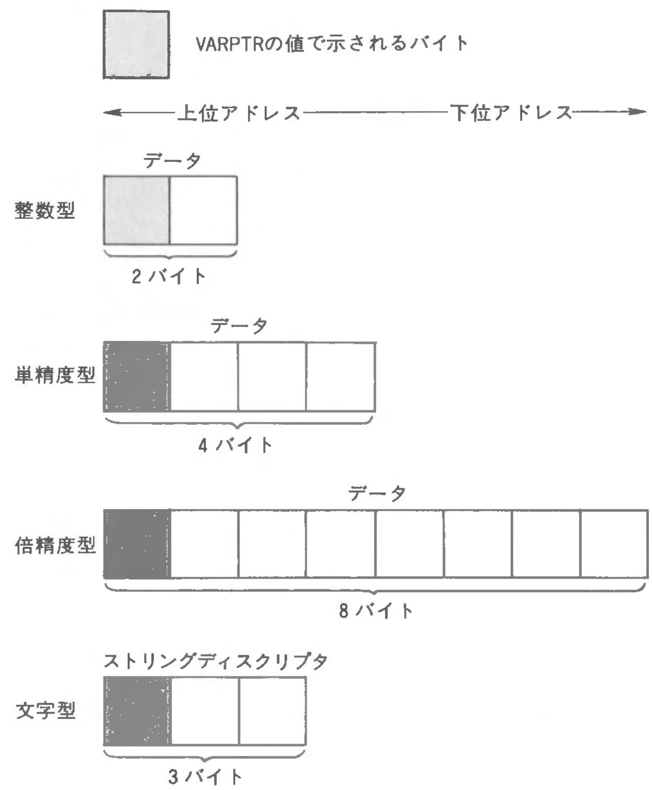
2）VARPTR（# <ファイル番号>）

文　例　1）PRINT HEX$（VARPTR（A$））

2）BSG%=VARPTR（#1）

解　説　1）<変数名>で指定した変数のデータが格納されている変数領域の先頭アドレスをデータセグメントのセグメントベースからのオフセット値で返します．

<変数名>で指定する変数はどんな型でもかまいませんが，VARPTR関数を使用する前にその変数には値が与えられていなければなりません．値の与えられていない変数にVARPTR関数を使用すると，"Illegal function call" エラーが生じます．



2）<ファイル番号>で指定したファイルに割り当てられているファイルバッファの先頭アドレスをデータセグメントのセグメントベースからのオフセット値で返します．

ファイルはVARPTR関数を使用する前に開かれていなければなりません．

サンプルプログラム

```
100 'VARPTRで得られる番地から
110 '                変数名を書き換えます。
120 A=123.456:B=654.321
130 APTR=VARPTR(A):BPTR=VARPTR(B)
140 ANAME=APTR-3:BNAME=BPTR-3
150 PRINT USING "  @ =###.### ";"A",A,"B",B
160 POKE ANAME,ASC("B")
170 POKE BNAME,ASC("A")
180 PRINT USING "  @ =###.### ";"A",A,"B",B
run
  A =123.456   B =654.321
  A =654.321   B =123.456
Ok
```

VARPTR$
関数

機 能　変数のアドレスを文字形式で返す.

書 式　VARPTR$ (<変数名>)

文 例　A$=VARPTR$(H$)

解 説　<変数名>で指定した変数のデータが格納されている変数領域のアドレスを，3バイトの文字形式で返します.

VARPTR$関数を実行する前に，指定した変数には値が与えられていなければなりません.
変数のアドレスは，次のような形式で与えられます．ただし，ここでのアドレスはデータセグメントからのオフセット値で与えられます.

バイト0	バイト1	バイト2
型	変数アドレスの下位バイト	変数アドレスの上位バイト

〝型″は〈変数名〉の型を示すもので，次のいずれかが与えられます.

2：整数型
3：文字型
4：単精度実数型
8：倍精度実数型

VARPTR$関数が返す値は，

```
CHR$ (<型>)+MKI$ (VARPTR (<変数名>))
```

と同様になります．したがって，

```
ASC (VARPTR$ (<変数名>))
```

とすれば〈変数名〉で指定した変数の型を得ることができ，

```
CVI (RIGHT$ (VARPTR$ (<変数名>), 2))
```

とすればVARPTR (<変数名>) と同じくその変数の先頭アドレスを得ることができます.

VARPTR$関数は，主としてPLAY文やDRAW文の中の変数名を示すのに使用します.
例えば，PLAY文で，

```
PLAY "XA$;"
```

という指定をしたとします．この形式をそのまま使用してもかまいませんが，VARPTR$関数を使

用して,

PLAY "X"+VARPTR$ (A$)

とすることができます.

サンプルプログラム

```
100 '下の4行の中1つを有効にしてください。
110 'DEFINT A:A=123
120 'DEFSTR A:A="ABC"
130  DEFSNG A:A=123.456
140 'DEFDBL A:A=1234.567890123
150 B=ASC(VARPTR$(A))
160 IF B=2 THEN PRINT "Aは整数です。":END
170 IF B=3 THEN PRINT "Aは文字列です。":END
180 IF B=4 THEN PRINT "Aは単精度実数です。":END
190 IF B=8 THEN PRINT "Aは倍精度実数です。":END

run
Aは単精度実数です。
Ok
```

VIEW

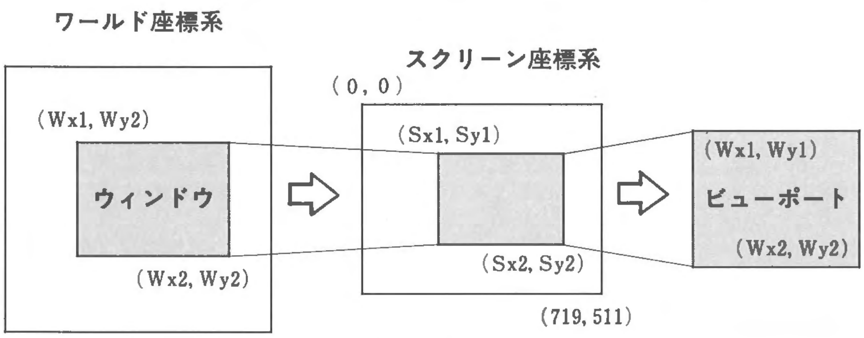
ステートメント

機 能	**ディスプレイ画面上での表示領域（ビューポート）を指定する.**
書 式	VIEW〔SCREEN〕〔(Sx1, Sy1)−(Sx2, Sy2)〔, <**領域色**>〕〔, <**境界色**>〕
文 例	VIEW (100, 30)−(200, 75),, 7
解 説	(Sx1, Sy1) を左上の頂点, (Sx2, Sy2) を右下の頂点とするスクリーン座標系上の長方形を図形表示領域として指定します.

この図形表示領域のことをビューポート（View Port）と呼びます.

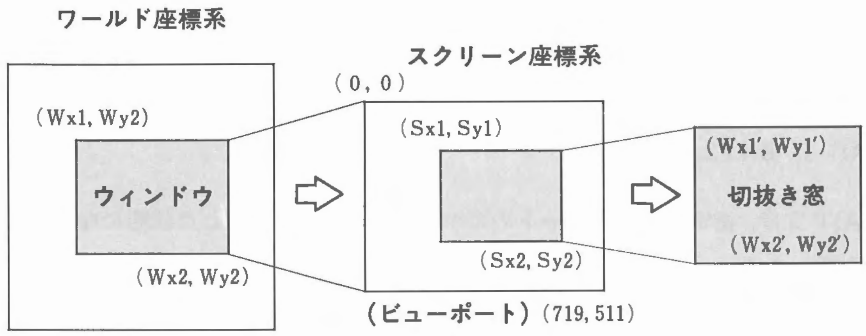
VIEW 文の実行以前に WINDOW 文が実行されていないか, WINDOW 文がパラメータなしで実行されていた場合, ビューポート内の座標系は, 左上の頂点を原点（0, 0）として各ピクセルと1対1に対応する座標系となります.

WINDOW 文がパラメータつきで実行されていた場合, ビューポート内の座標系は WINDOW 文で指定したウィンドウ内の座標系がその中に展開されるワールド座標系となります.



スクリーンオプションなしの場合

SCREEN オプションを付加した場合は, ディスプレイ画面全体がウィンドウとなり, ビューポートは単にそのウィンドウ内のワールド座標系の表示領域を限定する, 〝切抜き窓〟として働きます. すなわち, この〝切抜き窓〟以外のウィンドウ内に描かれる図形は画面上に表示されないことになります.



スクリーンオプションつきの場合

〈領域色〉が指定された場合には，指定されたパレット番号でビューポート内を塗りつぶします．

〈境界色〉が指定された場合には，指定されたパレット番号でビューポートの枠を描きます．

VIEW 文をパラメータなしで実行したときは，ディスプレイ画面そのものがビューポートを考えられる BASIC 起動時の初期状態と同じ状態になります．

VIEW 文は，図形の表示領域の指定を行うだけで，実際の画面に対する操作は行いませんので，VIEW 文によりビューポートを変更するだけで，以前のビューポート中の図形が移動するようなことはありません．また，VIEW 文はすべてのグラフィック画面表示の対象範囲を指定された領域に限定してしまいますから，ビューポートの外に点や線などを表示することはできません．

Sx1<Sx2, Sy1<Sy2 が成りたたない場合，あるいはこれらの座標がディスプレイ画面から外れている場合には，"Illegal function call" エラーとなります．

一度設定されたビューポートは，次に VIEW 文が用いられるまで変化しません．なお，VIEW 文が実行されると LP はビューポートの中央の点に移動します．

ウィンドウとビューポートについての詳細は，文法編および BASIC ユーザーズガイドをご覧ください．

参照　WINDOW，文法編　ウィンドウとビューポート

サンプルプログラム

```
100 SCREEN 1:CLS:KEY OFF :LOCATE 22
110 WINDOW(1,1)-(639,199)
120 GOSUB HAKO
130 VIEW (1,1)-(319,198),0,14
140 GOSUB HAKO
150 VIEW(500,1)-(630,99),0,13
160 GOSUB HAKO
170 END
180 HAKO
190  LINE(50,50)-(600,150),11,BF
200 RETURN
```

WAIT
ステートメント

機能　コンピュータの入力ポートをモニタする間，プログラムの実行を中断する．

書式　WAIT 〈ポート番号〉，〈式 1〉〔，〈式 2〉〕

文例　WAIT 1，&H 22，&H 22

解説　WAIT 文は，指定した入力ポートのビットパターンが，指定した状態になるまでプログラムの実行を中断します．

〈ポート番号〉で指定されたポートから読み込まれたデータはまず〈式 2〉との XOR がとられ，次にその結果と〈式 1〉の AND がとられます．もしその結果が 0（偽）なら，BASIC はもう一度ポートの状態を読み込み，同じ操作を繰り返します．もし結果が 0 以外（真）であれば，プログラムの実行は次の文に移ります．〈式 2〉が省略された場合は 0 とみなされます．

注意：WAIT 文の実行により無限ループにはいってしまう場合があります．その場合には PC-100 をリセットしなければなりません．

サンプルプログラム

```
100 ' WAIT
110 ' プリンタに文字を出力する
120 '
130 A$="Hello"+CHR$(13)+CHR$(10)
140 FOR I=1 TO LEN(A$)
150    ' ＢＵＳＹ信号が出なくなるまで待つ
160 WAIT &H14,&H20
170    ' １バイト出力
180 OUT &H12,ASC(MID$(A$,I)) XOR 255
190    ' ストローブラインをＯＮにする
200 OUT &H10,INP(&H10) OR 8
210    ' ストローブラインをＯＦＦにする
220 OUT &H10,INP(&H10) AND &HF7
230 NEXT I
```

```
Hello
```

WHILE～WEND

ステートメント

機能　条件が満足されている間，WHILE 文から WEND 文までの一連のステートメントやコマンドを繰り返し実行する．

書式　WHILE <論理式>
　∫
WEND

文例　WHILE J< 5
　∫
WEND

解説　<論理式> が真（0 でない）である間，WHILE～WEND ループ中にあるステートメントやコマンドを繰り返し実行します．

<論理式>を評価した結果が真であれば，WHILE～WEND ループ中の文を順番に実行し，WHILE 文に戻ります．この繰り返しは，<論理式> の評価が偽になるまで続けられ，<論理式> が偽になると，WEND 文に続く文に制御が移ります．最初に<論理式>を評価した結果が偽であれば，WHILE～WEND ループ中の文は一度も実行されません．

WHILE～WEND ループは FOR～NEXT ループと同じようにネスト（入れ子）構造にすることができます．この場合，1 つの WHILE～WEND ループは完全に他の WHILE～WEND ループあるいは FOR～NEXT ループの内部になければなりません．

なお，WEND 文は，WHILE 文と対になり，ループの終りを示す働きをしますので省略することはできません．

また，WHILE～WEND ループ中から GOTO 文などでループの外に出てしまうようなプログラムは，正しい動作が保証されなくなりますので，注意してください．

サンプルプログラム

```
100 I=1
110 WHILE I<6
120  PRINT I;"-";
130  J=1
140  WHILE J=<I
150   PRINT J;:J=J+1
160  WEND
170  PRINT:I=I+1
180 WEND
run
 1 - 1
 2 - 1  2
 3 - 1  2  3
 4 - 1  2  3  4
 5 - 1  2  3  4  5
Ok
```

WIDTH
ステートメント

機 能	デバイスやファイルに対して文字の領域を設定する.

書 式

1）WIDTH <桁数>〔, 行数>〕
2）WIDTH <ファイルスペック>, <サイズ>
3）WIDTH # <ファイル番号>, <サイズ>

文 例

1）WIDTH 80, 25
2）WIDTH "LPT1：", 80
3）WIDTH #1, 100

解 説

1）ディスプレイ画面に表示する文字の〈桁数〉と〈行数〉を決定します．画面が横表示か縦表示かにより，次のような組合せを指定できます．

横表示（桁×行）	縦表示（桁×行）
80×25 ＊	64×45 ＊
80×32 ＊	85×45
80×64	85×90
90×25 ＊	—
90×32 ＊	—
120×25	—
120×32	—
120×64	—

注）＊印は，2バイトキャラクタ（漢字など）の表示可能なモードであることを示します．このときのキャラクタフォントは8×16ピクセルです．

WIDTH 文によって指定された行数および桁数は，ディスプレイ画面上のキャラクタ座標系における行の最大値および桁の最大値となります．

上記のように組合せ以外の表示方法を，WIDTH USING 文を用いて指定することもできます．

なお，〈桁数〉あるいは〈行数〉を省略した場合は現在の桁数あるいは行数のままで変化しません．

2）〈ファイルスペック〉で指定されたデバイスに対してバッファのサイズを〈サイズ〉で指定された値に設定します．ここで許されているデバイスは，RS-232C コミュニケーションポート（COM 1：）とプリンタ（LPT 1：）です．

〈サイズ〉の値は0から255まで許されており，0は256と解釈されます．

例えば，このステートメントがプリンタに対して実行されたとすれば，プリンタの1行に印字される文字数は〈サイズ〉で指定したものとなります．このようにプリンタに対して実行した場合，WIDTH LPRINT と同等の機能を得ることができます．初期状態では255に設定されて

います.

3）〈ファイル番号〉で割り当てられているバッファ（チャネル）に対して，その大きさを〈サイズ〉で指定します．ここで許されているデバイスは，コミュニケーションポートとプリンタです．サイズは0から255までの値で0は256と解釈されます．以後このファイル番号を使っているデバイスとの入出力はこの〈サイズ〉単位で行われます．つまり，WIDTH文を実行すると，コミュニケーションポートやプリンタに対するデータの送出時に，指定された〈サイズ〉の位置ごとにキャリッジリターンコード＋ラインフィードコードを送出するようになるわけです．

なお，〈サイズ〉は，初期状態では255に設定されています．

参照 WIDTH USING, WIDTH LPRINT

サンプルプログラム

```
100 GOSUB KAKU
110 WIDTH 80,32:GOSUB KAKU
120 WIDTH 120,64:GOSUB KAKU
130 WIDTH 80,64:GOSUB KAKU
140 WIDTH 120,32:GOSUB KAKU
150 WIDTH 80,25:GOSUB KAKU:END
160 KAKU
170 CLS
180 PRINT "NEC PERSONAL COMPUTER PC-100"
190 PRINT "Push any key"
200 IF INKEY$="" THEN 200
210 RETURN
```

```
NEC PERSONAL COMPUTER PC-100
Push any key
```

WIDTH LPRINT

ステートメント

機 能 プリンタに出力される1行の表示文字数を設定する.

書 式 WIDTH LPRINT 〈文字数〉

文 例 WIDTH LPRINT 80

解 説 プリンタの1行に印字することのできる文字数を設定します. WIDTH"LPT1:", 〈文字数〉としても同じです. 〈文字数〉で指定することのできる値は, 0から255までで0は256と解釈されます.

参 照 WIDTH

サンプルプログラム

```
100 WIDTH LPRINT 30
110 FOR I=33 TO 131
120  LPRINT CHR$(I);
130 NEXT I
140 LPRINT CHR$(13)+CHR$(10)
150 END
```

```
!"#$%&'()*+,-./0123456789:;<=>
?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[¥
]^_`abcdefghijklmnopqrstuvwxyz
{|}~ _≠
```

WIDTH USING

ステートメント

機　能　ディスプレイ画面に表示する文字の大きさ，桁数，行数などを設定する．

書　式　WIDTH USING 〈フォント〉〔，〈桁スペース〉，〈行スペース〉〔，〈左マージン〉〕〔，〈右マージン〉〕〕

文　例　WIDTH USING 0，8，32，300，450

解　説　ディスプレイ画面に表示する文字の大きさ，キャラクタ座標系の形式を設定するためのステートメントです．

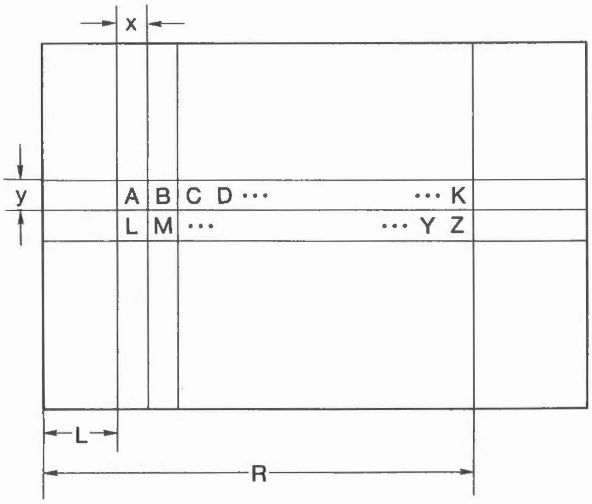
このステートメントは横表示の画面を設定するためのもので，縦表示の画面設定を行うことはできません．縦表示の画面モードとなっているときに WIDTH　USING 文を実行しますと，"Illegal function call" エラーが生じます．

〈フォント〉は 2 種類あり，次のように指定します．

0 ：フォントサイズ 8 ×16ピクセル
1 ：フォントサイズ 6 × 8 ピクセル

8 ×16ピクセルのフォントを指定したときのみ，2 バイトキャラクタ（漢字など）の表示が可能となります．2 バイトキャラクタのフォントサイズは16×16ピクセルとなります．

〈桁スペース〉，〈行スペース〉，〈左マージン〉，〈右マージン〉は次の図のように指定します．各パラメータに指定する値はピクセル単位で与えられます．



ただし，x ：〈桁スペース〉
y ：〈行スペース〉
L ：〈左マージン〉
R ：〈右マージン〉

WIDTH USING

〈桁スペース〉は，フォントのはいる〝マス〟の左右の大きさを表します．選択したフォントの横サイズ（8または6）～32の範囲の整数で指定します．

〈行スペース〉は，フォントのはいる〝マス〟の上下の高さを表します．選択したフォントの縦サイズ（16または8）～32の範囲の整数で指定します．

〈左マージン〉は，文字の表示を開始する左端の位置をスクリーン座標のX座標（0～719）で指定します．

〈右マージン〉は，文字の表示を折り返す右端の位置をスクリーン座標のX座標（0～719）で指定します．

WIDTH USING文は，キャラクタ座標系の形式を自由に指定することができますが，これはスクリーンエディタに対する指定ともなりますので，どのようなキャラクタ座標系を指定してもカーソルによる編集作業を行うことができます．

注意：WIDTH USING文を使用して画面の設定を行ったときは，SCREEN文によって漢字入力用テンプレートの表示をすることはできなくなりますので，ご注意ください．

参照　WIDTH

サンプルプログラム

```
100 GOSUB KAKU
110 WIDTH USING 1:GOSUB KAKU
120 WIDTH USING 1,32:GOSUB KAKU
130 WIDTH USING 0,32,32:GOSUB KAKU
140 WIDTH USING 0,16,16,300:GOSUB KAKU
150 WIDTH USING 0,8,32,300,450:GOSUB KAKU
160 WIDTH 80,25:END
170 KAKU
180 CLS
190 PRINT "NEC PERSONAL COMPUTER PC-100"
200 PRINT "Push any key"
210 IF INKEY$="" THEN 210
220 RETURN
```

```
N E C   P E R S O N A L   C O M P U T E R
P C - 1 0 0
P u s h   a n y   k e y
```

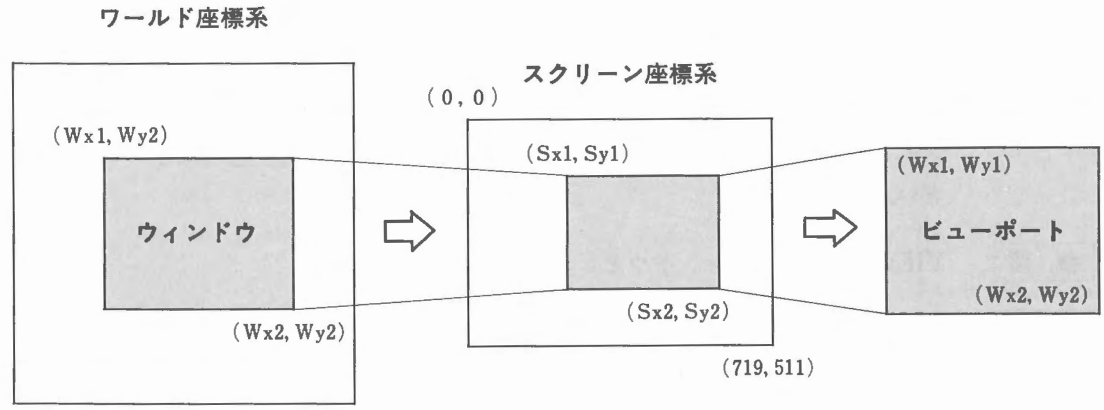
WINDOW

ステートメント

機　能	ビューポートに表示するワールド座標系内の領域（ウィンドウ）を指定する.
書　式	WINDOW 〔SCREEN〕〔(Wx 1, Wy 1)－(Wx 2, Wy 2)〕
文　例	WINDOW (－300, －50)－(100, 70)

解　説

(Wx 1, Wy 1) を左上の頂点, (Wx 2, Wy 2) を右下の頂点とするワールド座標系上の長方形で囲まれた領域を, ディスプレイ画面上のビューポートに表示する領域として指定します. 以後, この領域内に入る図形は VIEW 文で指定された領域 (ビューポート) 内に表示されるようになります. このようにして指定された領域は, ウィンドウと呼ばれ, ちょうどカメラのファィンダを通して景色を見たときのように, ウィンドウから外れた図形は表示されなくなります. また, ウィンドウの大きさを変えると, カメラのズーミングのように視野に入る領域の大きさが変り, ビューポートに表示される図形の大きさも変わります.



SCREEN オプションは, これをつけるとワールド座標系の Y 軸の方向とディスプレイ画面における Y 軸の方向とが一致し, ビューポート内は上から下に向かって Y 座標の値が増加するワールド座標系となります.

SCREEN オプションを省略すると, ワールド座標系の Y 軸の方向がディスプレイ画面における Y 軸の方向に対して反転するため, ビューポート内は下から上に向って Y 座標の値が増加する, 通常の数学座標（デカルト座標）表現のワールド座標系となります.

WINDOW 文をパラメータなしで実行すると, ワールド座標系に関しては BASIC 起動時の初期状態にもどり, ウィンドウのまったくない状態となります.

このときディスプレイ画面上に存在するものはスクリーン座標系のみとなり, ワールド座標系に対して働くグラフィックコマンドも, すべてスクリーン座標系に対するものとなります.

WINDOW

WINDOW 文も VIEW 文と同じように領域の指定を行うだけで，実際の画面に対する操作は行いませんので，WINDOW 文により，ウィンドウの設定を変更するだけで，ビューポート中に表示される図形が変化するようなことはありません．また，誤って，図形が何も描かれない領域をウィンドウとして設定すると，スクリーン上のビューポートには，何の図形も描かれませんので注意してください．

Wx 1，Wx 2，Wy 1，Wy 2 のとり得る値の範囲は－9. 99999E＋62～9. 99999E＋62 ですが，WINDOW 文で指定できる座標の範囲は，｜Wx 2－Wx 1｜≦9.99999E＋62，｜Wy 2－Wy 1｜≦9.99999E＋62 です．

Wx 1<Wx 2, Wy 1<Wy 2 が成り立たない場合，あるいはこれらの座標がワールド座標系から外れている場合には "Illegal function call" エラーとなります．WINDOW 文で指定する座標は，ワールド座標ですから，負の値や実数値もとれます．

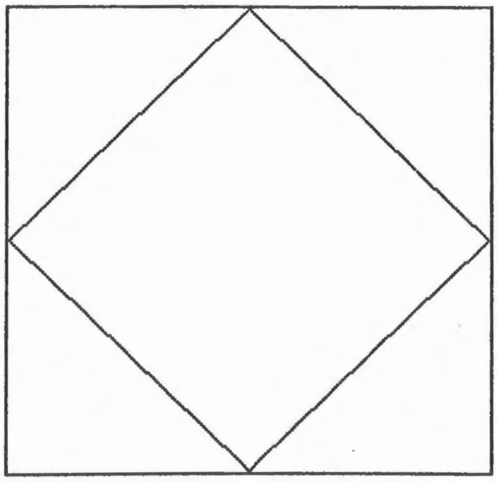
一度設定されたウィンドウは，次に WINDOW 文が用いられるまで変化しません．なお，WINDOW 文が実行されると LP はウィンドウの中央の点に移動します．

ウィンドウとビューポートについての詳細は，文法編および BASIC ユーザーズガイドをご覧ください．

参照 VIEW，文法編　ウィンドウとビューポート

サンプルプログラム

```
100 SCREEN 1:CLS:LOCATE 1,1,0
110 C=9
120 VIEW(200,50)-(400,250),,7
130 WINDOW(-100,-100)-(100,100)
140 GOSUB *DRA
150 WINDOW(-100,-100)-(0,0)
160 GOSUB *DRA
170 WINDOW(0,0)-(100,-100)
180 GOSUB *DRA
190 WINDOW(-500,-500)-(500,500)
200 GOSUB *DRA
210 END
220 *DRA
230  LINE(0,100)-(100,0),C:LINE -(0,-100),C+1
235   LINE -(-100,0),C+2:LINE -(0,100),C+3
240  FOR J=0 TO 1000:NEXT J
250   CLS
260 RETURN
1230    LINE -(-100,0),C:LINE -(0,100),C
```



WRITE
ステートメント

機能	ディスプレイ画面へデータを出力する.
書式	WRITE 〔〈式〉〕{ ; , } ・・・〕
文例	WRITE I * J, K, A$

解説

〈式〉で表される数値，文字列などのデータをディスプレイ画面に出力します．〈式〉が省略された場合には改行のみを行います．

式は，カンマ（,）あるいはセミコロン（;）により区切ります．カンマとセミコロンとの機能上の区別はありませんが，画面上には，各式の区切りとして必ずカンマが表示されます．BASIC は，各式の値を出力したのち，改行を行います．

WRITE 文は，ほぼ PRINT 文と同じ様に式の値の出力を行いますが，不要なスペースは詰められ，それぞれの式の値の間は必ずカンマで区切られます．また文字列はダブルクォーテーション（"）で囲まれて出力されます．

参照 PRINT, WRITE#

サンプルプログラム

```
100 A%=123
110 B!=9.87654E+31
120 C#=3.14159265359#
130 D$="N100-BASIC"
140 PRINT A%;B!,C#;D$
150 WRITE A%;B!,C#;D$
```

```
run
 123  9.87654E+31              3.14159265359 N100-BASIC
123,9.87654E+31,3.14159265359,"N100-BASIC"
Ok
```

WRITE#

ステートメント

機能 シーケンシャルファイルへデータを書き出す．

書式 WRITE #〈ファイル番号〉〔,〈式〉{ ; | , } ・・・〕

文例 WRITE #1, I * J, K, A$

解説 〈式〉で表される数値，文字列などのデータを，〈ファイル番号〉により指定されたファイルに書き出します．

〈ファイル番号〉は，ファイルを出力モードでOPENしたときに指定した番号です．〈式〉は，カンマ（,）あるいは，セミコロン（;）によって区切ることができます．カンマとセミコロンとの機能上の区別はありません．

WRITE #文は，ほぼPRINT #文と同じ様に式の値を出力しますが，不要なスペースは詰められ，それぞれの式の値の間は必ずカンマで区切られます．また文字列はダブルクォーテーション（"）で囲まれて出力されます．

WRITE #文は，各式の値を書き出した後，キャリッジリターンコード＋ラインフィードのコードを書き出します．〈式〉が省略された場合には，単にキャリッジリターンコード＋ラインフィードコードのみが書き出されます．

WRITE #文は，区切り記号としてカンマを必ず出力し，また不要なスペースの出力は行わないため，PRINT #文に比べファイルの使用領域を節約することができます．

参照 PRINT#, WRITE

サンプルプログラム

```
100 A$="ABCDEF":B$="GHIJKLMN"
110 PRINT "ｶｷｺﾝﾀﾞ DATA ﾊ  ";A$;" ﾄ ";B$:PRINT
120 OPEN "TEST8" FOR OUTPUT AS #1
130 WRITE#1,A$;B$
140 CLOSE
150 PRINT "WRITE : ";:GOSUB YOMIKOMI
160 OPEN "TEST8" FOR OUTPUT AS #1
170 PRINT #1, A$;B$
180 CLOSE
190 PRINT "PRINT : ";:GOSUB YOMIKOMI
200 END
210 YOMIKOMI
220 OPEN "TEST8" FOR INPUT AS #1
230 LINE INPUT#1,Z$
240 CLOSE
250 PRINT Z$
260 RETURN
```

WRITE #

```
run
ｶｷｺﾝﾀﾞ DATA ﾊ  ABCDEF ﾄ GHIJKLMN

WRITE : "ABCDEF","GHIJKLMN"
PRINT : ABCDEFGHIJKLMN
Ok
```

N100-BASIC
REFERENCE MANUAL

付　録

1　予約語

ABS
AND
ASC
ATN
AUTO
BEEP
BLOAD
BSAVE
CALL
CALLS
CDBL
CDBL $
CHAIN
CHDIR
CHR $
CINT
CIRCLE
CLEAR
CLOSE
CLS
COLOR
COM
COMMON
CONSOLE
CONT
COS
CSNG
CSNG $
CSRLIN
CVD
CVI
CVS
DATA
DATE $
DEF
DEFDBL
DEFINT
DEFSNG
DEFSTR
DELETE
DIM
DRAW
DSKF
DSKI $
DSKO
DSKO $
EDIT
ELSE
END
ENVIRON
EOF
EQV
ERASE
ERDEV
ERL
ERR
ERROR
EXP
FIELD
FILES
FIX
FN
FOR
FRE
GET
GO TO
GOSUB
GOTO
HEX $
IF
IMP
INKEY $
INP
INPUT
INPUT #
INPUT $
INPUT¥
INSTR
INT
IOCTL
IOCTL $
JIS
JIS $
KEY
KILL
KLEN
KPOS
KTN
KTN $
LCOPY
LEFT $
LEN
LET
LINE
LIST
LLIST
LOAD
LOC
LOCATE
LOF
LOG
LPOS
LPRINT
LSET
MERGE
MID $
MKD $
MKDIR
MKI $
MKS $
MOD
MOTOR
MOUSE
NAME
NEW
NEXT
NOT
OCT $
OFF
ON
OPEN
OPTION
OR
OUT
PAINT
PALETTE
PEEK
PEN
PLAY
PMAP
POINT
POKE
POS
PRESET
PRINT
PSET
PUT
RANDOMIZE
READ
REM
RENUM
RESET
RESTORE
RESUME
RETURN
RIGHT $
RMDIR
RND
RSET
RUN
SAVE
SCREEN
SGN
SHELL
SIN
SOUND
SPACE $
SPC (
SQR
STEP
STICK
STOP
STR $
STRIG
STRING $
SWAP
SYSTEM
TAB (
TAN
THEN
TIME $
TIMER
TO
TROFF
TRON
USING
USR
VAL
VARPTR
VIEW
WAIT
WEND
WHILE
WIDTH
WINDOW
WRITE
WRITE #
XOR

2 エラーメッセージ

エラーメッセージ	コード	意　　　　味
Advanced Feature	73	将来のために予約されている．
Bad file name	64	ファイル名の指定がまちがっている．
Bad file number	52	オープンしていないファイルや起動時に指定していないファイルを参照した．
Bad file mode	54	シーケンシャルでオープンしたファイルに対してランダムアクセスをしようとした．またはその逆．
Bad record number	63	レコード番号の指定が誤っている．
Can't continue	17	CONT コマンドによって実行を再開することができない．
Communication buffer overflow	69	周辺機器との入出力のためのバッファがオーバーフローした．
Dead lock	77	システムエラー．
Device Fault	25	デバイスドライバにエラーが生じた．
Device I/O Error	57	デバイスがI/Oエラーを起こした．
Device Timeout	24	一定時間待ったがデバイスからデータが来なかった．
Device Unavai lable	68	サポートされていないデバイスを使おうとした．
Direct statement in file	66	アスキー形式のファイルを読み込む際にダイレクトステートメントが存在した．
Disk full	61	ディスク上に書き込むスペースがないのに書き込もうとした．
Disk media error	72	メディアのタイプがちがう．
Disk not Ready	71	ディスクが用意されていない．
Disk write protected	70	ディスクにライトプロテクトシールが貼ってある．
Division by zero	11	0による割り算が実行されて，その結果の値がオーバーフローした．
Duplicate Definition	10	配列またはユーザー関数を二重定義しようとした．
Duplicate label	33	同じ名前のラベルが2つ以上存在している．
FIELD overflow	50	FIELD文においてファイルバッファ以上の大きさの領域を指定した．

エラーメッセージ	コード	意　　味
File already exists	58	NAME文によって変更しようとしたファイル名が既に存在している.
File already open	55	既にオープンされているファイルに対してOPEN, KILLなどを実行した.
File not found	53	LOAD, SAVE, KILLなどで現在のディスクに存在しないファイルを指定した.
FOR without NEXT	26	FOR～NEXTが正しく対応していない（FORが多い）.
Illegal direct	12	ダイレクトモードで使用できない文を実行しようとした.
Illegal function call	5	ステートメントおよび関数において，機能の呼び方が誤っている.
Input past end	62	ファイル中の全てのデータを読み尽した後に，さらに入力文が実行された.
Internal error	51	BASIC内部にエラーが生じた.
Line buffer overflow	23	1行で入力できる文字の範囲を超えて入力が行われた.
Missing operand	22	ステートメント中必要なパラメータが指定されていない.
NEXT without FOR	1	FOR～NEXTが正しく対応していない（NEXTが多い）.
No RESUME	19	エラー処理ルーチンの中にRESUME文がなく，プログラムの実行が継続できない.
Out of DATA	4	読むべきデータがないのにREAD文が実行された.
Out of memory	7	メモリ容量が足りなくなった（プログラムが大きすぎる，またはスタックを消費し尽した）.
Out of paper	2	プリンタ用紙がなくなった.
Out of string space	14	文字列を格納するメモリ領域がなくなった.
Overflow	6	演算結果や入力された数値が，許される範囲を越えた.
Path File Access Error	75	指定したディレクトリはすでに存在している.
Path not found	76	指定したパスがみつからない.
Rename across disks	74	NAME文において，異なるドライブ間でリネームしようとした.
RESUME without error	20	エラー処理ルーチンに制御を移さなかったのにRESUME文がある.

エラーメッセージ	コード	意　　味
RETURN without GOSUB	3	GOSUB～RETURNが正しく対応していない. (RETURNが多い)
String formula too complex	16	文字式が複雑すぎる. (カッコのネスティングが深すぎるなど)
String too long	15	1つの文字変数内の文字数が255文字を超えている.
Subscript out of range	9	配列変数の添字の値が規定の範囲から外れている.
Syntax error	2	文の記述が文法通りになっていない.
Too many files	67	ファイルの数が許容される範囲を超えた.
Type mismatch	13	式の式辺・右辺, 関数の引数などにおいて変数の型が一致していない.
Undefined label	34	定義されていないラベルで参照した.
Undefined line number	8	飛び先として必要とされる行番号 (GOTO, GOSUBなどの飛び先) が存在しない.
Undefined user function	18	DEF FN文によって定義されていないユーザー関数を引用しようとした.
Unprintable error	未定義	メッセージの定義されていないエラー.
WEND without WHILE	30	WHILE～WENDが正しく対応していない. (WENDが多い)
WHILE without WEND	29	WHILE～WENDが正しく対応していない. (WHILEが多い)

3 デバイス名

デバイス名	機器名称	入 力	出 力	備　　考
A：	フロッピーディスク	○	○	Model-10では1台のディスクユニットA：,B：を兼用.
B：	フロッピーディスク	○	○	
C：	RAMディスク	○	○	SETUPコマンドで設定する.
D：	ハードディスク	○	○	PC-98H31またはPC-98H33
E：	ハードディスク	○	○	PC-98H32またはPC-98H34
COM1：	RS-232Cポート	○	○	
KYBD：	キーボード	○	×	
LPT1：	プリンタ	×	○	
SCRN：	スクリーン	×	○	

NEC